Investigating the Effectiveness of Problem Templates on Learning in Intelligent Tutoring Systems

BSc Honours Report

Moffat Mathews[1]
Supervisor: Dr. Antonija Mitrović[2]
{m.mathews[1] | tanja[2]}@cosc.canterbury.ac.nz
Computer Science & Software Engineering
University of Canterbury
6 November 2006

Abstract

Deliberate practice within a coached environment is required for skill acquisition and mastery. Intelligent Tutoring Systems (ITSs) provide such an environment. A goal in ITS development is to find means to maximise effective learning. This provides the motivation for the project presented.

This paper proposes the notion of *problem templates*. These mental constructs extend the idea of memory templates, and allow experts in a domain to store vast amounts of domain-specific information that are easily accessible when faced with a problem. This research aims to examine the validity of such a construct and investigate its role in regards to effective learning within ITSs.

After extensive background research, an evaluation study was performed at the University of Canterbury. Physical representations of problem templates were formed in Structured Query Language (SQL). These were used to model students, select problems, and provide customised feedback in the experimental version of SQL-Tutor, an Intelligent Tutoring System. The control group used the original version of SQL-Tutor where pedagogical (problem selection and feedback) and modelling decisions were based on constraints. Preliminary results show that such a construct could exist; furthermore, it could be used to help students attain high levels of expertise within a domain. Students using template based ITS showed high levels of learning within short periods of time. The author suggests further evaluation studies to investigate the extent and detail of its effect on learning.

Acknowledgements

The author would like to thank his supervisor, Tanja Mitrovic for her inspiration, encouragement, and guidance throughout this project. Many thanks to Stephanie DeMay for proof reading, and trying to make sense of what was intended to be said in this report. To all the fellow ICTG members, especially Jay Holland, Nancy Milik, Kon Zakharov, and Amanda Nicholas who provided much help and discussion during the year, thank you! Finally, to all the honours students: it has been a pleasure going crazy with all of you. May the gods be gracious and restore you all to your former mental health and well-being.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 The Importance of Learning

Learning and instruction have been studied in various fields such as Psychology, Education, Neuroscience, Artificial Intelligence, and more recently Intelligent Tutoring Systems. Learning is defined as the process leading to relatively permanent or potential behavioural change; it is the increased likelihood of a response to a stimulus as a consequence of experience [1]. It affects the way we perceive a particular environment, the way we interpret incoming stimuli, and consequently how we interact or behave in a given situation. As a result, learning is an imperative part of practically every aspect of our lives. Furthermore, most sectors of our society, be it businesses, governments, military organisations, or individuals, are striving to increase the effectiveness of learning in an attempt to increase productivity and efficiency, achieve goals (organisational or individual), and provide and retain a competitive edge. Various learning theories, such as behaviourism, cognitivism, and constructivism, have been proposed to try to understand the processes involved in learning. Research into the effects of learning on the brain and neurological systems also continue in an aid to study learning from a physiological perspective [2-4].

1.2 Problems in the Education System

Although learning is of great importance in our society, our education systems are fraught with problems.

The methods we use to facilitate learning are far from perfect. Some researchers even claim that there is a crisis in the education field and that education as a whole is on a decline [5]. Irrespective of their position on the state or trend of education today, researchers seem to agree on how little we know about the complex processes and interaction between teaching techniques, the content, the learning process, and the learner [6].

Funding issues, amongst other factors often lead to high student-to-teacher ratios [7] and inadequate resources. A large proportion of the teacher's role, energy, and time is now devoted to being a disciplinarian, organiser, and even peacemaker, rather than primarily focussing on teaching. In an attempt to tailor the course work to a large group of students, the teacher is compelled to aim the course content and set the speed of progress at the 'average' student level, thereby providing the ideal level for only a small percentage of students. With high student numbers, pupil interaction is low. The teaching process turns into one of knowledge transfer, where declarative knowledge is transferred from the teacher to the passive student. The role of practice with adequate feedback is denigrated. Deep learning however, requires the student to play an active role, involving high participation and interaction between the student and the teacher. Practice is an essential part of skill acquisition [8].

Most models used in education systems today view both the student and content as static entities [6]. These models force all students to trace the same path through a pre-defined curriculum at a pre-set pace. However, neither students nor content are static entities. Students have differing initial knowledge levels. They also have differing capabilities that vary between sub domains. This means that the ideal learning environment should be customised to each individual student. Each student's entry position into the course material should be at a position appropriate to his or her knowledge level. They should also be able to follow their own path through the content at their own pace dependent on their current and updated level.

Psychological studies show that students could learn and retain more information from discovery than from direct instruction [9]. The school system has been criticised for removing the ability for students to learn by discovery; instead, it forces students to learn by memorising facts prescribed by the curriculum. A call to re-evaluate the education system from an epistemological point of view and empower the student's ideas thereby providing opportunities for discovery is given in [10]. However, it has also been noted that the amount learned from unguided exploration or deviation from a set path is usually proportional to expertise. In other words, experts could benefit from learning through exploration, whereas novices could get overwhelmed and lose direction with unguided exploration [7]. This implies that learning should be continually monitored and customised to the knowledge level and expertise of the student; providing more guidance, direction, and feedback for the novice and allowing greater freedom for exploration and choice as expertise increases. This level of monitoring and customising is only possible in either one-to-one human tutoring or with Intelligent Tutoring Systems (ITSs), the latter being more practical.

1.3 Motivation for the Research

ITSs are computer-based, interactive, adaptive, learning environments that allow students to practice domain-specific skills whilst receiving explicit feedback to their solutions. Each student's knowledge of various portions of the domain is recorded in a student model. From this, a knowledge level is continually calculated and updated. Both the model and the knowledge level are used to customise and create each student's path through the course material. ITSs shift the focus from knowledge transfer to knowledge communication [7]; the student is no longer a passive recipient of knowledge. Instead, they are actively involved in the learning process that has been adapted to their current knowledge and capabilities. ITSs provide a good solution to the problems mentioned in Section 1.2.

Although ITSs have been used successfully in a number of domains achieving high learning rates within short periods [11-13], the learning rates are still not as high as one-to-one human tutoring [14]. A main goal of ITS development is to increase the effectiveness of learning within the ITS. It is this goal that provides the impetus for this research. Two main methods of achieving this goal exist. First, modules within the ITS can be enhanced to increase effectiveness. Second, conjectures regarding various aspects of learning can be made and evaluated. This research attempts both, with greater emphasis on the latter.

1.4 An Overview of the Research

How do experts differ from novices? De Groot [15] was one of the first to try and fathom how chess experts were able to find the best move from within a vast range of possibilities. Researchers have found that instead of searching through a tree of all possible moves as novices do, chess masters are able to recognise chess configurations and attach meaningful representations to these patterns helping them plan ahead [8]. It is as if they had developed a library of configurations or patterns or templates. This has been supported by Chase and Simon's Chunking Theory [16, 17] and more recently the Template Theory of Memory (TT) [18].

The TT proposes that experts store domain information in *templates* in long-term memory. Templates, like chunks, are groups of meaningfully related information. Templates contain slots that either store domain information (such as recognisable patterns of problem states, or solutions) or pointers to other templates. Pointers to templates are held in short-term memory, thereby enabling almost instantaneous retrieval of vast amounts of information with the same short-term memory constraints. This information allows the expert to quickly recognise the problem state, associate common solution strategies, and even predict future paths. According to the TT, experts are those that have formed a mental library of templates. This enables experts to recognise certain types of problems, and link relevant problem-solving information to the template.

From personal observation, templates are used in a wide range of domains in human tutoring to enable novices to learn problem solving skills and associate techniques with particular parts of a problem. For example, when learning to drive, the student creates and adds to a set (mental library) of templates, which aids in solving a particular problem. These templates include *hill starts*, *three point turns*, *parallel parking*, *uncontrolled intersections*, etc. Cues to recognise the template in a problem, and strategies or techniques to solve the problem are stored in the template slots. When presented with a problem (e.g. drive from university to home), the student accesses the relevant templates, and uses the techniques associated with them to solve the overall problem. This reduces the number of items the student holds in working memory, and provides a quicker, more efficient method of accessing related 'chunks' of information. The human driving instructor maintains a student model based on the student's knowledge of the template, and is able to generate appropriate problems containing templates that require further practice.

We hypothesise that when learning on an ITS based on templates, non-experts, who already have basic instruction in the domain, would be able to learn more effectively. More specifically, students should be able to recall greater amounts of domain knowledge while still having high rates of learning.

To evaluate this hypothesis, an evaluation study was conducted. Templates for the chosen domain, Structured Query Language (SQL) were compiled. Two versions of a Constraint-Based ITS (SQL-Tutor [19]) were implemented: a control version in which constraints were used to make pedagogical and modelling decisions, and an experimental version using templates. Participants for the study were students from an undergraduate database course who used the system for the second term in 2006. Data such as pre and post-tests, student models, and logs were recorded and later statistically analysed; learning curves were plotted. A detailed account of the research is given in this report.

1.5 The Structure of the Report

Section 2 of this report looks at relevant background information related to this project. It overviews ITSs, theories of memory, and notions on how experts differ from novices. It is not an exhaustive list of the extensive background knowledge required to comprehend this field. However, it contains references to articles, providing the reader with links to further research if required.

The goals and hypothesis of the research are outlined in Section 3. Problem templates and existing examples are described in Section 3.3. Section 4 contains the design and implementation of the research study, while the description of the evaluation is given in Section 5. A discussion also follows the reporting of the results. Finally, a few probable extensions to the research are proposed after the conclusion in Section 6.

2 Background

2.1 Intelligent Tutoring System (ITS)

Many factors accentuate the need for ITSs in learning situations. Three of the main factors are described below.

First, one-to-one human tutoring provides the optimal learning environment [14]. However, due to many factors including resource and practical constraints, most academic institutions fall short of this optimum with high student-to-teacher ratios [5]. ITSs are a solution to this problem as they provide a learning environment that can be customised to each individual by maintaining an updated student model. The ITS also allows the student to learn at their own pace without being restricted by that of the class.

Second, practice is a central part of learning and skill acquisition [20]. Regular, optimised, deliberate practice within a structured (coached) environment is required to maximise learning and attain mastery [8]. Ideal learning scenarios emphasise this by presenting the learner with a *target task* that is part of a customised *training sequence* and presented within a *task environment* suited for the particular domain-dependent context. The target task should always be within the individual's zone of proximal development [21] i.e. difficult enough to be challenging, yet not too difficult or too easy to be de-motivating. The training sequence should be continually adapting to the individual’s current and updated knowledge level, such that no two students trace the same path through the system. The task environment should not only present the tasks and solution space in such a manner as to keep the cognitive load on the student at a manageable level, but also present the context of the particular task. Here again, the ITS is the ideal solution as it presents the student with tasks based on the individual’s student model in the appropriate task environment.

Third, “rewards and punishments ... are crucial ingredients of learning, as is contact with, and manipulation of, the environment” ([1] p.565). This implies that for effective learning to occur, the student needs to have precise feedback at the correct level of detail, adapted to the student’s solution. It also re-states the need for hands-on practice in the correct context. The impracticality of this (usually due to high student-to-teacher ratios) in a *normal* classroom situation makes ITSs the ideal solution to this need. Various levels of feedback or forward hints can be given at either user-requested or ITS- specified points during the task. This feedback could be adaptive (dependent on the student’s knowledge) and scaffolded (where the detail of feedback presented is inversely proportional to the student’s level of expertise) [22].

2.1.1 Architecture of an ITS

The structure of an ITS is often divided into four main components: the student model, the pedagogical module, the domain knowledge module, and the communication module. Often a fifth component, the expert module, is also included [23]. Figure 2 shows the overall architecture and relationships between the components in the ITS.

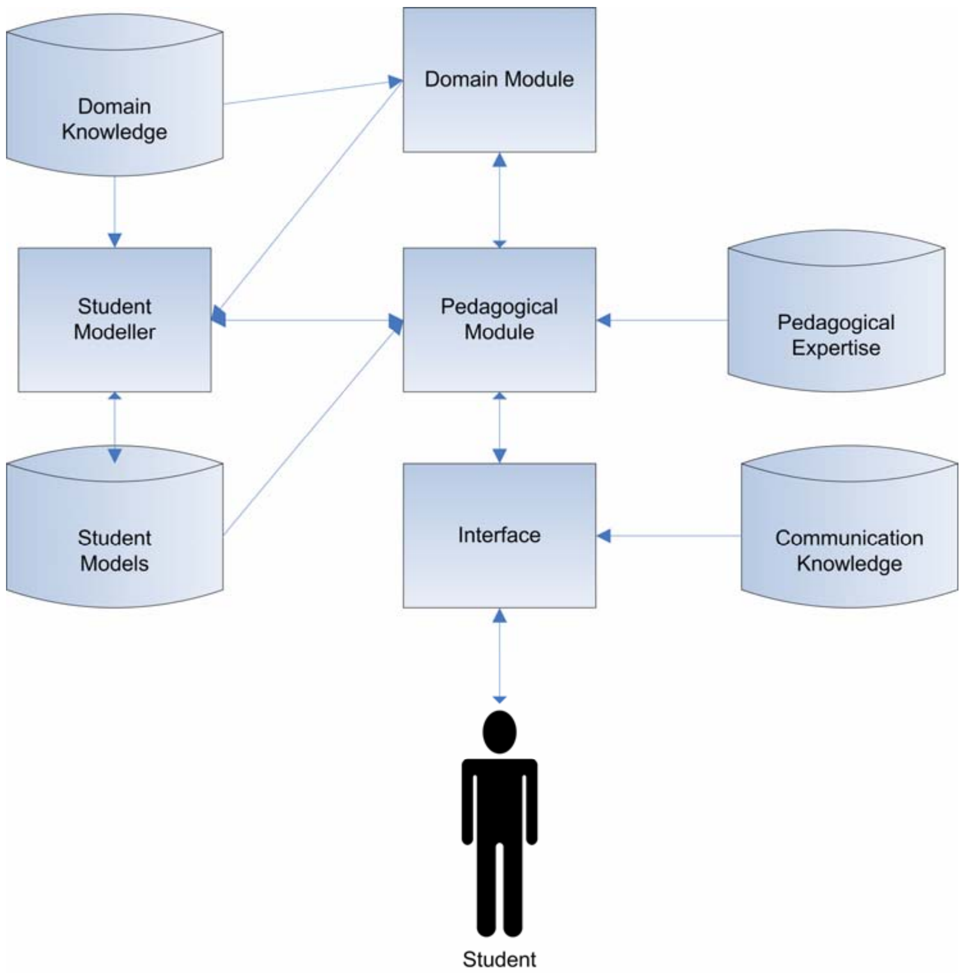


Figure 1: The architecture of an ITS

"The student model is a representation of the computer system's beliefs about the learner and is, therefore, an abstract representation of the learner in the system." [24] The student model contains an estimation of the level of domain (or sub domain) mastery for each student. It is this model that the system utilises to make pedagogical decisions that are customised to each student. For instance, a long-term student model containing the student's entire history on the ITS can be used to govern high level decisions such as problem selection, while a short-term student model can be used to guide decisions regarding feedback on the current solution. What information should be stored in a student model has been long questioned. Self in [25] talks about the intractability of building detailed and precise student models. He suggests that the ITS needs not possess such detailed models to be able to tutor the student satisfactorily. For instance, student models in a Constraint-Based Modelling (CBM) tutor could hold problem histories of both attempted and solved problems, constraint histories, current feedback levels, and the student's current step in the current problem. Two popular modelling techniques, Model Tracing (MT) and CBM, are described in Section 2.1.2. The student modeller evaluates student solutions and dynamically updates the student models.

The pedagogical module is responsible for the *tutoring* decisions made by the ITS i.e. what to instruct, and how to support the student. It usually performs these decisions using information from the individual student models. These decisions are commonly reflected in problem selection and feedback. For problem selection, the pedagogical module employs strategies to select the *next best (ideally suited) problem* for the student. The strategies implemented can vary between ITSs and are generally in line with educational strategies chosen by the designers of the system. For feedback, decisions regarding when and how to intervene, the level of detail of feedback, and the degree of scaffolding are made. For instance, MT tutors generally provide feedback at every step of a student's solution, interrupting the student's incorrect progress and forcing adherence to the correct solution at every step; whereas CBM tutors generally provide feedback on submission of a problem. ITSs can employ various strategies for level of feedback. Strategies can range from manual selection, where the student chooses the level required, to a fully automated level, where the ITS decides the level based on factors such as the student model. Strategies for scaffolding feedback, or reducing feedback as the expertise increases, can also be implemented.

The domain module contains domain knowledge, problems, and optionally associated ideal solutions. Domain knowledge can be represented in various ways such as production rules in MT tutors (e.g. 600 rules in the ANDES tutor [26]) and constraints in CBM tutors (e.g. 81 constraints in NORMIT tutor [27]). In some instances, the domain module can contain an expert system or problem solver that is able to generate the correct solution according to the path chosen by the student (e.g. [28, 29]).

The communication module contains both the interface and optionally, a stored representation of the communication knowledge. The interface is the means by which the student communicates with the ITS. It presents the problem, associated information, and solution space within the task environment. A goal of the interface is to present the information in such a manner that minimises the cognitive load on the student. Interfaces can also contain different tools such as calculators, diagram constructors, and equation lists [26]. Help (both domain help and ITS-specific help) can be made available to the student via the interface. The communication knowledge contains strategies to communicate adequately with the student. For instance, students with high spatial ability could benefit from communications that include pictorial representations; furthermore the dual Cognitive Theory of Multimedia Learning states that certain students could benefit from various multimedia representations of communication e.g. the dual channels assumption [30].

2.1.2 Modelling in ITS

Two main types of modelling approaches are common in ITSs today: Model Tracing (MT) and Constraint-Based Modelling (CBM). The evaluation study was performed on SQL-Tutor, a CBM tutor. For a comparison of the techniques, see [31]. A brief description of both approaches is given below.

2.1.2.1 Model Tracing

The MT technique is a short-term student modelling approach derived from the ACT-R [9] cognitive theory (formerly the ACT* theory [32]). This theory makes a clear distinction between declarative and procedural knowledge, and outlines three learning phases: the acquisition of declarative knowledge, knowledge compilation, and strengthening. The acquisition of declarative knowledge is the transfer of domain-related facts to the student. Knowledge compilation is the transformation of the declarative knowledge into procedural knowledge. It occurs when problem-solving behaviour is created by relating declarative knowledge to task goals i.e. by practice. It is seen as the most important phase of learning. During this process, production rules (rule-based representations of procedural knowledge) are formed. It is this step that MT targets. Finally, the strengthening of both declarative and procedural knowledge occurs with practice.

MT utilises production rules for each step of a procedure. A production rule is a goal-oriented if-then rule that consists of a *goal*, a *situation* or context in which this rule is applied, and an *action* required to take the student from the current situation to the required goal. The production rule is of the form:

> If the required goal is <goal>
> and the current situation is <situation>, then
> perform <action>.

An example of a production rule could be:

> If the required goal is to have "light in a room",
> and the current situation is "the light switch is turned off", then
> perform the action "turn the light switch on".

An MT tutor contains production rules to generate paths to correct solutions that the designer has implemented. The MT tutor also contains production rules to generate possible incorrect actions that the student could make while solving the problem, and stores them as a library of *buggy rules*. The libraries of correct and buggy production rules are not exhaustive; they merely signify the procedures that the designer of the system has implemented. They might not allow for novel and creative correct solutions entered by the user, or incorrect actions unanticipated by the designer. At each step of the solution procedure, the student's rule is matched against the set of rules applicable for that particular goal and situation. If the matching rule is correct (i.e. the appropriate action for that situation to reach the goal was taken), the student is allowed to continue; if not, the procedure is interrupted and a feedback message (from the buggy rule that was matched) is presented to the student. See [33, 34] for more information on the MT approach and associated ITSs.

Long term student models in cognitive tutors are implemented using the Knowledge Tracing (KT) approach [33, 35]. Estimates of each student's knowledge of production rules are held as probabilities in the model. While MT determines the immediate feedback received on the current step within the problem procedure, KT controls the path taken by the student through the course material by governing problem selection.

2.1.2.2 Constraint-Based Modelling

CBM [36] is a student modelling approach proposed by Ohlsson and is based on the theory of learning from performance errors [37]. According to this theory, students learn using two cognitive functions: error detection and error correction. Detecting the error requires domain-specific, declarative knowledge, while correcting the error requires the student to modify their internal rule set. An ITS should model what is correct in the domain (declarative knowledge), detect any errors made by the student, and allow novel and creative solutions. The space of false knowledge is very large; it is much greater than the space of correct knowledge [38]. Modelling domain knowledge using the bug library system, the machine learning approach, or the model tracing technique severely restricts the capability of both the ITS and the student in the learning environment (see [36] for a comparison of the three techniques).

Ohlsson proposed a technique whereby domain knowledge is modelled in the form of a set of constraints. Constraints define sets of equivalent problem states; they are pedagogically equivalent and trigger the same instructional action [38]. Each constraint is an ordered pair *(C_r, C_s)*, where C_r is the relevance condition and C_s is the satisfaction condition. C_r identifies the problem states in which the constraint is relevant. C_s asserts additional conditions that must be satisfied or true for the problem state to be correct. A constraint therefore is of the form:

> "If <relevance condition> is true then
> <satisfaction condition> had better also be true,

otherwise something has gone wrong." [29]

An example of a constraint in the domain of fraction addition could be:

> If the denominators in the problem are different, then
> it better be that the denominator of the solution is the lowest common denominator, otherwise something has gone wrong.

If the student's solution state is the same as the relevance condition, then it must also be in the same state as is described by the satisfaction correction for it to be correct. Often, a feedback message (or set of messages) can also be included with every constraint to present on violation of the constraint. Using this technique, a student can be modelled according to their knowledge and usage of constraints.

CBM is an effective and efficient means of modelling both the domain and the student within an ITS. It does not require the compilation of a bug library, a task that necessitates extensive studies into student errors. It can be used in a variety of domains ranging from structured to open-ended. Using CBM, modelling is reduced to a series of simple pattern matching procedures. Patterns can be then represented in compiled forms such as RETE networks [38].

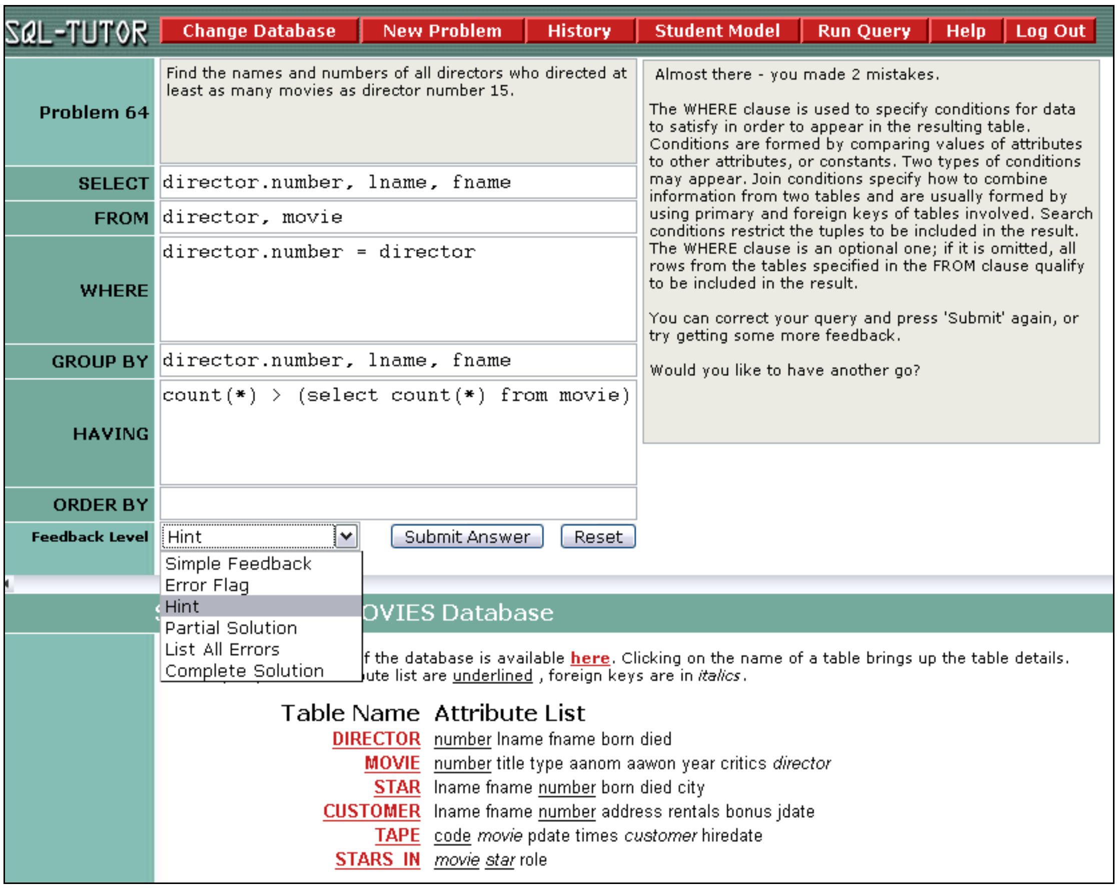
2.1.3 SQL-Tutor[1]

SQL-Tutor [7, 19] is a web-based, CBM ITS developed and maintained by the Intelligent Computer Tutoring Group[2] at the University of Canterbury. Since its conception in 1996, SQL-Tutor has undergone many changes and evaluations (see [39, 40] for more information). It provides the student with a customised problem-solving environment in the Structured Query Language (SQL) domain. The assumption is that SQL-Tutor is used to complement classroom instruction for students who have undergone direct instruction in SQL.

SQL is a relational database standard which evolved from Structured English Query Language (SEQUEL) [41], a language developed for System R at the IBM Research Labs in San Jose in the early 1970's. It became an ANSI standard in 1986, and an ISO standard the following year. SQL contains three components: a data definition language (DDL) for manipulating database objects, a view definition language (VDL) for defining views, and a data manipulation language (DML) for manipulating data within the database. Although there could be multiple solutions for the same problem, SQL is a very structured language that conforms to set rules and specifications. SQL-Tutor focuses on the database querying, perhaps the largest and most used component of SQL. See [42] for a more thorough look at SQL.

Currently, SQL-Tutor has approximately 300 problems, each belonging to one of 13 databases. The databases provide the student with contextual information regarding the problem. In the task environment [see Figure 2], the student is presented with the problem text and the database schema information. Additional information about each relation, such as attribute names, description, and types can be viewed by clicking on the table name. From within the ITS, students can query a sample database with their solution and view the results of their query. An ideal solution for each problem is also stored in the domain module. As the domain is modelled using constraints, all correct variations of the solution are accepted by the ITS.

[1] SQL-Tutor available at http://ictg.cosc.canterbury.ac.nz:8000.
[2] ICTG: http://ictg.cosc.canterbury.ac.nz.



Problem 64

Find the names and numbers of all directors who directed at least as many movies as director number 15.

SELECT	director.number, lname, fname
FROM	director, movie
WHERE	director.number = director
GROUP BY	director.number, lname, fname
HAVING	count(*) > (select count(*) from movie)
ORDER BY	
Feedback Level	Hint

Simple Feedback
Error Flag
Hint
Partial Solution
List All Errors
Complete Solution

Almost there - you made 2 mistakes.

The WHERE clause is used to specify conditions for data to satisfy in order to appear in the resulting table. Conditions are formed by comparing values of attributes to other attributes, or constants. Two types of conditions may appear. Join conditions specify how to combine information from two tables and are usually formed by using primary and foreign keys of tables involved. Search conditions restrict the tuples to be included in the result. The WHERE clause is an optional one; if it is omitted, all rows from the tables specified in the FROM clause qualify to be included in the result.

You can correct your query and press 'Submit' again, or try getting some more feedback.

Would you like to have another go?

OVIES Database

f the database is available **here**. Clicking on the name of a table brings up the table details. ute list are underlined , foreign keys are in *italics*.

Table Name	Attribute List
DIRECTOR	number lname fname born died
MOVIE	number title type aanom aawon year critics *director*
STAR	lname fname number born died city
CUSTOMER	lname fname number address rentals bonus jdate
TAPE	code *movie* pdate times *customer* hiredate
STARS_IN	*movie* *star* role

Figure 2: The task environment in SQL-Tutor

Students can enter their solution into the solution space provided and submit it at any stage. Upon submission, the individual student model is updated, and feedback specific to the task is provided in the feedback pane. Six levels of feedback exist: Simple feedback, error flag, hint, partial solution, list all errors, and full solution (in ascending order of detail). The feedback level automatically increases on each submission to a maximum of *hint*. Any of the feedback levels can be manually selected by the student. See [22, 39, 43] for more information on various strategies of problem selection and feedback that have been used within SQL-Tutor.

Student knowledge is modelled using constraints. SQL-Tutor utilises an open student model [44, 45] which can be displayed in both a textual and graphical form. The current session history, showing problem attempts can also be viewed at any time within each session.

The architecture of SQL-Tutor is similar to the general architecture of an ITS, and is shown in Figure 3. The databases, problems, and their associated ideal solutions are held in the domain module. The SQL domain is modelled using a set of approximately 700 constraints. Each constraint has a feedback message associated with it. On violation of that particular constraint, the feedback message could be displayed depending on the pedagogical strategy chosen. Individual student models are stored for each student. The student modeller evaluates student solutions, and updates the student models. The pedagogical module invokes pedagogical strategies for both problem choice and type of feedback, and with the student model provides a customised training sequence. Information such as problem attempts, time, student, and ITS decisions (e.g. problem choice), level of feedback, etc are recorded in individual student logs. The session manager is responsible for maintaining each online learning session. It also enables the handling of states in HTML, a stateless protocol. Finally, SQL-Tutor is run on the AllegroServe[3] web server, and is made available to students via the Internet. Students can log into SQL-Tutor using their web browser. In-depth information regarding the SQL architecture can be found in [19].

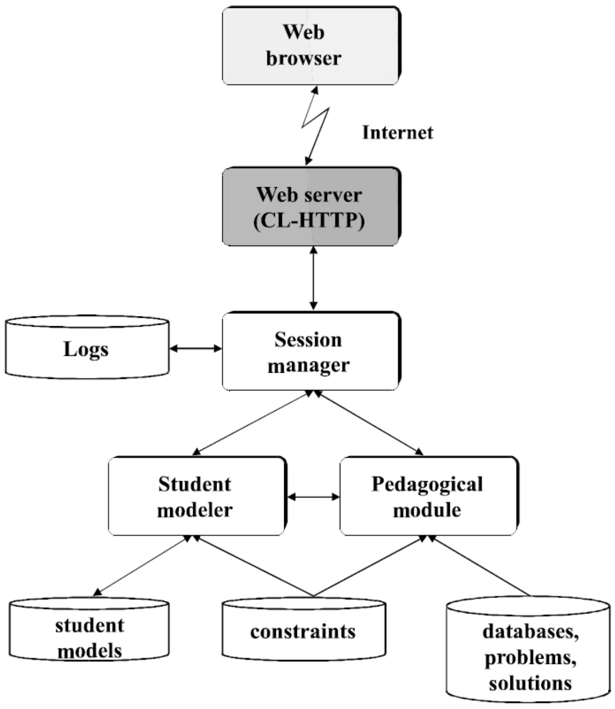


Figure 3: Architecture of SQL-Tutor [19].

2.2 Learning, Memory, and Expertise

Hippocrates (c. 460 BC) was the first to propose that both thought and knowledge were housed in the brain. Today we know that the cerebral cortex supports short-term memory while the long-term memory involves the limbic system [1]; recent experiences and learning alter parts of the neocortex [46, 47]. Research into learning has followed two, sometimes intertwined paths, to create theories and models of learning. The first looks at learning externally from a behavioural and psychophysical point of view. The second views the internal mechanisms and physiological changes in the brain due to learning, using technology such as PET[4], MRI[5], or CAT[6] scans. The research described in this section primarily reflects the first path; that of creating models and theories based on experimental observations on the learner, the content, and the environment or context. More specifically, we look at how experts view their domain, how they learn, and how they use domain knowledge to solve common problems within their domain. Theories such as molecular theory (investigates how RNA and protein molecules are changed within the neurons due to learning) and morphological theory (investigates changes in the relationship between cells) have been constructed under the second path of research. See [1, 48] for information on the second path.

2.2.1 Theories of Expert Memory

The ability of experts to memorise a significantly greater amount of domain-related information than non-experts, termed the *expertise effect in memory recall,* has been

[3] Available from Franz Inc.: http://www.franz.com.
[4] Positron Emission Tomography.
[5] Magnetic Resonance Imaging, formerly Magnetic Resonance Topography (MRT) or Nuclear Magnetic Resonance (NMR).
[6] Computed Axial Tomography or Computed Tomography (CT) Imaging.

the subject of much research and debate in the field of psychology [49]. De Groot [15] was the first to report this effect in 1946. Since then, a number of process and ecological theories have been proposed to account for this phenomenon. These theories often encompass information from a variety of fields such as cognition, memory, expertise research, learning, epistemology, etc. Although much of the research was done in the domain of chess, the expertise effect has also been observed in other domains such as games, mnemonics, music, sciences, and sports [50-53]. The remainder of this subsection provides an overview of the relevant theories.

In his experiments involving problem-solving tasks in the domain of chess, De Groot found no significant differences in the types of search patterns employed by players, such as depth of search, number of moves considered, or the heuristics used. However, when briefly presented with domain-specific information, experts were vastly superior at memorising this information than non-experts. Experts were also able to handle a far greater amount of domain information. Since then, research has shown that experts demonstrate superior performance at tasks that require the ability to memorise, recall, and retrieve vast amounts of domain-specific data. For example, [54] analyses the memory skill of a waiter who could take up to 20 dinner orders without notes. In [50], after two years of training and practice, a subject was able to memorise and recall up to 82 digits that were read aloud; with no training, most humans are able to recall 7 digits [55]. It was initially believed that this was due to certain individuals possessing exceptional processing and cognitive skills. However, this notion has been dismissed after further investigation revealed that experts largely have the same cognitive and memory constraints as non-experts [17]. (See [56] for an argument that working memory is an important component of general fluid intelligence).

There are two main memory storage areas: short-term memory (STM), and a long-term storage area. Although the exact size of STM is still largely unknown, it is accepted that its capacity is relatively small; between 2 to 9 chunks [53, 55, 57]. Long-term memory (LTM) has a very large capacity, and is much more durable than STM [58]. Most cognitive psychologists today also allow for an area with limited capacity that is distinct from the LTM, called working memory (WM) which was originally proposed in [59]. The WM contains a complex set of interacting subsystems used for short-term recall.

For many years, Chase and Simon's influential *chunking theory* of memory [17] was used to explain the expertise effect. This theory proposes that memorising and recall involves the use of two main elements: a *chunk* and a *pointer*. A chunk is a group of logically related information seen by the individual as a single unit and is stored in the LTM. It is a group of elements that have strong links or associations to each other, but weak links to elements within other chunks [60]. A pointer (or label) to this information takes up one slot in the STM. According to this theory both the expert and novice have 7±2 slots in STM (as suggested by Miller [55]). However, the novice has each slot filled with a single item of data, whereas the expert has pointers to each chunk in LTM. This gives the expert the ability to encode information rapidly by merely adding pointers to the STM. It also allows the expert access to a far greater amount of information almost instantaneously, such as identification of a good move in chess [53]. Chase and Simon proposed that chess experts have over many years, stored a large number of specific patterns of chess plays or chunks in LTM. They also found that the effect of expertise was eliminated when chess boards with randomly arranged pieces were used in their recall experiments; novices and experts performed similarly on recall in these experiments.

Chunking can occur in two ways. The first is a deliberate process, consciously implemented by the learner. This is called *goal-oriented chunking*. The second is an automatic chunking mechanism that occurs during perception called *perceptual chunking* [60].

A modification of the theory was given by Ericsson and Kintch (the LT-WM theory), that involves two components of working memory: a short-term working memory (ST-WM), and a long-term working memory (LT-WM). The ST-WM is a temporary storage area that contains pointers to information stored in the LT-WM, a more durable area of working memory [58]. It postulates the presence of schemas and patterns to aid in memorisation and recall.

A number of partitioning experiments were also conducted. These experiments showed that subjects were quicker at tasks involving grouped or partitioned clusters. It also showed that experts had larger groups in all levels of partitioning, and the group size increased as the positions became more typical [53].

TT [18] is an extension of the chunking theory. It suggests that information is hierarchically organised into chunks; chunks can recursively contain sub-chunks. They are also indexed by a hierarchical discrimination network. More frequently occurring chunks are converted to higher-level structures called *templates*. These templates are stored in LTM, while pointers to them can be held in STM. Each template can have many slots. Each slot can contain either related information or a pointer to another template. Learning domain information takes the same amount of time as specified by the chunking theory however, rapid LTM encoding of the slots in the templates occurs [53]. Due to the fact that the template slots can be filled quickly and that templates can contain other templates recursively, the amount of information able to be recalled as a single *chunk* can be very large. This allows experts over time to attach a large amount of relevant information to each problem state. In chess, this allows Grand Masters to have enough information to recognise, memorise, and recall a particular game board and even produce all the possible moves from that point forward. A comparison of theories is given in [51].

2.2.2 Points from the Theories of Expert Memory Relevant to this Research

Although each theory differs subtly, common points exist that are relevant to this research.

First, domain knowledge not cognitive ability defines expertise. Second, humans use patterns and schemas to organise domain information. Meaningfully related items are grouped together into partitions. Furthermore, experts have larger sizes of partitions i.e. they are able to link more related items together. Third, experts store information regarding problem states in recursively contained chunks or templates. Chunk size is proportional to the level of expertise. These templates contain information to recognise the problem state, and common strategies to *solve* the current problem. This not only allows for quick recognition of the state and almost instantaneous recall of solutions, but also allows for prediction and planning. Fourth, learning information takes time; however, encoding of information into template slots can occur rapidly. Templates are usually built with practice in the domain. Lastly, chunks either can be deliberately and explicitly created or learned automatically and implicitly.

3 Hypotheses and Goals of this Research

3.1 Goals

Six main goals were identified for this research. These were to:

1. introduce the notion of problem templates; conduct a thorough review of related research to examine the possible validity of this notion,
2. explore the plausibility of compiling physical representations of templates in a domain; specifically tailored for use in ITSs,
3. model students using problem templates as knowledge components within an ITS,
4. investigate the formation of problem selection strategies using information about problem templates,
5. assess the ability to provide template-based feedback, related to problems within an ITS, and
6. analyse the effects of template-based pedagogical decisions on learning by conducting an evaluation study.

3.2 Hypotheses

In line with our goals, our hypotheses were as follows:

1. That the notion of problem templates and their usefulness in learning situations would be supported by related research and current examples of use,
2. that physical representations of problem templates could be created within domains,
3. that students could be modelled using problem templates. Furthermore, pedagogical strategies such as problem selection and feedback could be based on problem templates,
4. that with an adequately long learning period, the learning rates would be higher in the template-based ITS, and
5. that after a sufficiently long period of practice, the recall of applicable domain-specific knowledge would also be higher in a template-based ITS than those in a non template-based ITS.

We propose that although templates take effort and time to compile, associate, and learn, teaching students "expert knowledge" in the form of problem templates would enable them to reach higher levels of expertise in shorter amounts of time.

3.3 Problem Templates

Problem templates, as defined in this paper, are chunks of domain-specific knowledge, compiled mentally by experts, and used to solve commonly occurring problems in a particular domain. We believe that problem templates are an extension to memory templates as proposed by the TT. These templates are therefore stored in LTM with pointers referring to them in STM. This paper proposes that within problem templates, experts potentially hold three types of information. First, problem templates contain information to *recognise particular problem states*. This is seen in the many chess experiments described in prior research. Second, they contain *common strategies to solve* the problem (i.e. take the user from the current state to an intended solution state). Finally, a list of *tools and associated information* required to implement these solutions could be included. As with memory templates, problem templates can also hold pointers to other templates. Because of these templates, experts have access to vast amounts of domain-specific information applicable to the current context. They are able to almost instantaneously recognise problem states within their domain, and seemingly effortlessly implement solutions.

In [61], Shanteau reviews a variety of research to describe many traits of experts. Amongst these, there are a few interesting themes to consider. First, expertise is domain-specific i.e. an expert is only an expert in his or her domain. Second, experts use heuristics to solve problems. As expertise increases, they use less deduction-based thinking and more pattern-matching type strategies. Third, experts are able to quickly retrieve a large amount of domain-specific information. They not only keep up-to-date with declarative information, but also are able to access relevant information to quickly alter strategies for various situations including exceptional circumstances. Experts even associate irrelevant information that can sometimes be used in decision making, to their detriment. Fourth, they are able to break complex problems into simpler tasks and utilise common strategies to solve them using a *divide and conquer* approach.

We believe that the themes specified in the paragraph above, support experts' use of problem templates. First, problem templates are domain-specific. Second, experts utilise pattern-matching strategies to recognise the problem state, and implement appropriate solution strategies. In other words, experts use information in problem templates. Third, the TT suggests that experts can hold vast amounts of data using templates, where slots within each template can recursively contain pointers to other templates. Because the expert is dealing with lightweight pointers rather than large amounts of data, it is easier for them to quickly alter their strategy by retrieving another pointer. This way they can account for exceptional or varying circumstances. Categorising information into separate chunks is dependent on the expert; templates could easily contain pointers to irrelevant data, which would be accessed when the template is accessed. This could be detrimental in making decisions. Fourth, when a complex problem is encountered, the template retrieved could recursively have its slots filled with pointers to other templates, each of which hold information to solve portions of the original problem. Just the simple procedure of retrieving the templates breaks complex problems into simpler ones. The process of retrieving and implementing the strategies for each template would automatically be utilising the *divide and conquer* approach.

Research into expert performance suggests that while novices reason backwards from a goal, experts proceed by forward reasoning (cited in [8]). This enables experts to not only plan, but also to anticipate future moves or problem states. We believe that problem templates explain this phenomenon. The hierarchical nature of templates creates a tree-like structure; where internal templates predominantly carry pointers and leaf-node templates carry associated data. Using these pointers, the internal structure of the tree is relatively lightweight, and thus easy to traverse. From the root node, it is easy to follow any of the paths leading from pointers to templates to predict future problem states. Given a tree of possible paths and states, the expert is able to progress by strategically planning ahead. As novices do not have this structure created in memory, they have to envision the goal, and create paths back to the current problem state. Because they are dealing with data rather than pointers, their cognitive load is very high, increasing the time and effort required to solve the problem. The lack of associated templates severely restricts the novice's ability to plan.

In certain domains, unnecessary, high risk-taking behaviour that causes severe safety concerns has been attributed to inexperience. As the level of experience increases, this behaviour seems to decrease rapidly. For instance, in the driving domain, road safety is often compromised by inexperienced drivers engaging in high risk-taking behaviour. See [62] and [63] for differences in this behaviour between novices and experts in the fields of driving and aviation respectively. The theory of problem templates could go a long way in explaining this behaviour. Novices have a very limited template set. This severely restricts their ability to predict or plan future scenarios. The high cognitive load described in the previous paragraph, also decreases their capability to envision probable future consequences of present actions, thus eliciting high-risk behaviours. Programmes such as driver education are created in an effort to increase experience; or in our view, impart template information in the form of coached, expert training.

The key to gaining expert levels of performance is deliberate practice [8, 64]. This practice has to be goal-oriented, focused, active, of long duration (usually many years), and under the supervision of an expert teacher or coach [64]. In the context of problem templates, the reasons for this seem clear. Not only do the three steps for learning [see Section 2] need to be traversed, but templates also need to be compiled. Information such as declarative knowledge and compiled procedural knowledge needs to be encoded into templates and stored in LTM. Pointers to these templates also need to be stored in the associated templates. Experts require successful and fast retrieval of appropriate information applicable to the task. Thus, the adequate placement of these pointers would then determine the level of expertise. The placement of these pointers could occur through experience or through coaching by teachers who are experts in the domain.

3.4 Examples of Existing Templates

As discussed earlier, learning affects nearly every aspect of our life. As such, examples of problem templates and their use are readily visible in our daily life; a few of which are given below.

In this simple example, we look at a common task that is often taken for granted; that of compiling a route between two frequently visited places, such as between one's workplace and home. When the need for this route is first established (e.g. a new job), one has to learn the various routes between work and home. This not only includes the declarative knowledge to get there (e.g. from looking at a map), but also the procedural knowledge (from practice). Associated information for each street or part of the route is also learned. This includes the distance, speed that one can travel, best or worst time of day to use this route, the flow of traffic, etc. Strategies to solve common problems that are related to that street or part of the route (for instance, traffic congestions during school drop-off times or very busy intersections) are also stored within the same template. Once these templates are compiled mentally, driving from home to work at any hour of the day becomes an almost automatic behaviour. Pointers to various associated templates make it easy to choose adjoining streets (or parts of routes) while still continuing in the intended direction. Whilst novices on the route might flounder with problems (either common or exceptional), experts effortlessly traverse the route. They manoeuvre through traffic, alter the route as necessary (using related pointers), and are able to plan ahead (e.g. expecting and avoiding route-related events), without even thinking about it. Problem templates give experts the ability to manipulate vast amounts of information using heuristics and patterns. This ability becomes second nature to such an extent that the expert could have very little conscious knowledge of the drive. This is seen when people keep driving back to their old home on “auto pilot” after having moved house.

We believe that another example of using problem templates occurs in practical driving instruction. When teaching driving, two methods could be employed: teach the precise technical details of driving, or teach using what we believe to be problem templates. Using the first method, driving instruction would consist of teaching technical values for variables for different driving *topics*. For instance, the beginner might learn the various RPM[7] values and ranges for the various vehicle velocities, at different vehicle weights and angles of inclination, etc., for each gear change in a manual shift. Devices such as accelerometers could be installed on the dashboard to aid in precise driving.

The second method of instruction seems much less precise and technical; in fact it seems to be based on patterns and heuristics created by experts. Instructors select certain commonly occurring high-level problems, and present common solution strategies, that have over time, been compiled by experts. Students are taught to *recognise the problem state,* apply *common strategies to solve the problem*, and select and use the appropriate *tools*. We see these as being problem templates. In the driving domain, these templates are named: hill-start, three-point turn, uncontrolled intersection, parallel parking, etc. As an example, when learning to do a hill-start (compiling the hill-start problem template), the student is taught to recognise the problem state which demands the need for the hill-start solution. They are then instructed in the appropriate common strategy[8] for solving this problem. The tools required, and their correct usage is also taught. This could involve the correct use of gears, indicators, etc. The instructor then gives complex problems such as “drive from work to home”, requiring the use of many associated templates.

Although the first method seems to be more specific and theoretically complete, it is surprisingly not used; it was created by the author for comparison. It is the second method that is taught as standard practice. After learning about problem templates, one can easily understand why.

Many examples of problem templates can be found in sport related domains. The ability of grandmasters in chess as described in research indicates the use of problem templates. The domain of golf utilises problem templates extensively. At any problem stage (e.g. the sand bunker, green, fairway, etc), the expert is able to immediately recognise the state, select the correct tool (e.g. sand wedge, putter, driver - irons/woods), and implement the correct strategy (e.g. wedge shot, putt, drive).

The field of Object-Oriented (OO) software design has embraced the idea of compiling strategies that experts use to solve common problems. These strategies are in the form of patterns. These patterns were most influentially described by Christopher Alexander in the architectural domain [65, 66]. Patterns such as the *Gang of Four Design Patterns* [67] have influenced the way in which novices in the field of software engineering are taught. In this context, patterns are reusable strategies, compiled by experts, for solving commonly occurring problems in software design. Since then, analysis patterns [68] have looked at strategies from a higher-level domain perspective (such as accounting, medicine, etc), rather than from a low-level implementation perspective. These patterns allow students to recognise particular problem states that are common to their domain, and use the strategies described to solve the problems. They also contain associated relevant information such as pointers to other patterns.

Collaborative problem templates are also found in domains such as sports. These are problem templates that are shared amongst individuals, with the purpose of solving a problem collaboratively. Game-plays (pattern-plays) for instance, are collaborative strategies used by members of a sports team to recognise a particular

[7] RPM: Revolutions Per Minute
[8] http://www.johnfoote.co.uk/manouevres/hillstarts.htm

game state, implement a specified strategy to *solve* the problem, list the usage of the *tools* (equipment), and associated information to aid in the solution. Problem templates allow these strategies to be very complex, contain changeable elements (such as player positions), and be altered dynamically as necessary. Game-plays are common in most team sports, such as basketball[9], hockey[10], football, etc.

[9] http://www.guidetocoachingbasketball.com/stations.htm
[10] http://www.jes-soft.com/hockey/downplay.html

4 Design and Implementation

4.1 The Choice of Domain and ITS

To conduct this research, an appropriate domain had to be chosen. An ITS also had to either be built or modified to include the new components of the system. The domain chosen was SQL, and the decision was made to modify an existing tutor rather than build a new one. The reasons for the decisions are listed below.

The main goals of this research was to see if physical representations of templates could be created, if pedagogical decisions could be based on them, and if students' domain knowledge could be modelled using them. Furthermore, there was a requirement to be able to harvest sufficient expert domain knowledge to compile templates. SQL is a rich domain, with a sufficiently wide range of different concepts making it a good choice for constructing a variety of templates for our research.

Reasons for choosing SQL-Tutor as the ITS for evaluation, are listed below. First, this research required an evaluation study to be performed whereby students in both the control and experimental groups could use the tutor for adequate periods in realistic learning situations. The ICTG maintains ITSs that are used in certain undergraduate courses at the University of Canterbury. Using one of these tutors for the study provided the setting to conduct the evaluation study in a real learning environment. Second, building an ITS (which includes modelling the domain and creating problems and solutions) requires time and expertise in the domain. Moreover, the creation of the ITS does not add anything to this research. Due to this reason, it was perspicacious to opt for modifying an existing tutor that was to be used in the undergraduate course. Third, SQL-Tutor contains a large library of problems; a requirement for creating the template sets. Finally, SQL-Tutor is the second-to-last tutor used in the course, and unlike the other sub domains taught in the course, is followed by a lab test in SQL. This not only ensured time for implementation, but also provided an incentive (the lab test) for students to make adequate use of the tutor.

4.2 The Creation of Templates

Ideally, templates for the most common problems would be compiled by a group of experts in the particular domain. This could take years of refining and perfecting the templates, creating associated links and patterns, and categorising templates involving much discussion and debate. See Ward's Wiki[11] for a similar process applied to the field of Object Oriented Software Engineering. Unfortunately, neither

[11] Ward's Wiki: http://urchin.earth.li/cgi-bin/twic/wiki/view.pl?page=WardsWiki

the level of expertise nor the time required for the process described was available for this research.

Thus, the creation of templates was based on an assumption: that the expertise required for this research was embedded in the domain module of SQL-Tutor; more specifically in the problem and ideal solution sets. SQL-Tutor contains problems and associated ideal solutions written by experts since 1996, specifically for the students in an undergraduate database course. This could mean that the problem set contains the most common problems in SQL for that range of student expertise. Furthermore, that the solution strategies to fill the slots of the template are found in the set of ideal solutions.

Once this assumption was made, various strategies were used to find correlations between problems and between solutions. Categorising was done both programmatically (using functions we created) and manually.

In the grouping strategy implemented, problems were recursively categorised into various groups, and then sub groups, using a different criterion on each pass. Criteria chosen ranged from level of nested statements in the solution, clauses contained in the solution, the number of relations and associated attributes involved, types of attributes involved (such as primary key, foreign key, non-key attribute, etc), aggregate functions, keywords, and similarity of conceptual actions on the database (e.g. a join on two tables can either be done in the *from* clause, or the *where* clause), etc. All solutions were evaluated against each criterion such that on each pass, one criterion was evaluated; problems and associated solutions having the same result (i.e. they were the same with respect to the criterion) were grouped into the same category.

For example, the *level of nested statements* criteria would categorise queries 1 and 2 shown below into the same group (level of nesting = 1), and query 3 into another (level of nesting = 2).

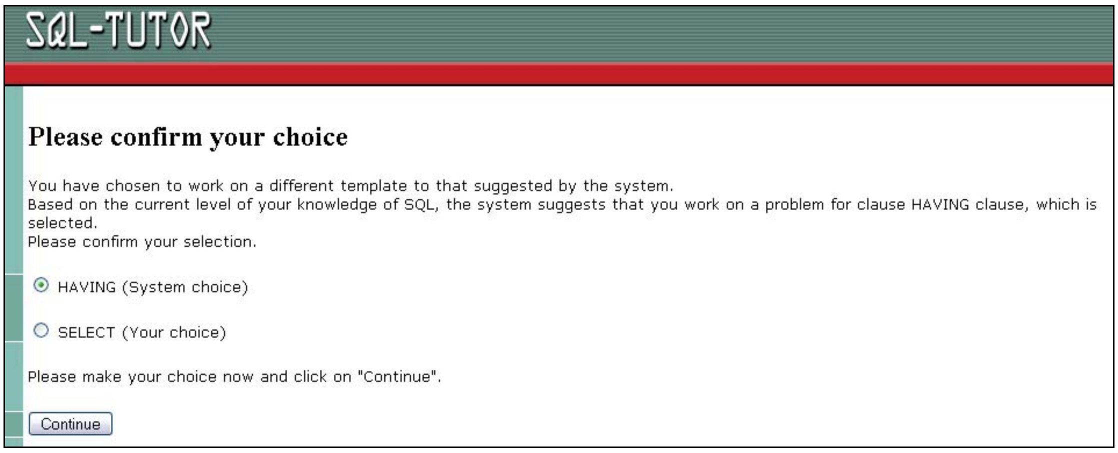
```
1.	select surname from customer
2.	select account_type and customer.name
	from account, customer
	where customer.id = account.customer
3.	select director.number, lname, fname
	From director join movie on director.number=director
	Group by director.number, lname, fname
	Having count (*) >= (select count (*) from movie where
	director=15)
```

On completion of the passes, the problems and solutions were classified into 38 groups, whereby each solution within the group had the same overall structure either in physical structure or in conceptual logic (i.e. the end result of the query would yield the same results). The next step was to create one generic solution for each category. To do this, relations and associated attributes were replaced by generic variables as shown in examples below. Statement 1 describes a relation and generically names it rel1. Statement 2 describes a regular attribute for the relation generically named rel1. Statement 3 describes a primary key attribute for the relation generically named rel2. Statement 4 describes a foreign key attribute for the relation generically named rel1.

```
1.	<rel1>
2.	<rel1.att1>
3.	<rel2.pk>
4.	<rel1.fk>
```

At this stage, we were left with 38 generic statements representing common solution strategies labelled as ideal solutions by experts, for all the problems in SQL-

Tutor. To complete the formation of the templates, these statements were sequentially numbered, and a feedback message attached to each. This feedback message would be shown to the student on violation of the template. Examples of templates are shown in Appendices.


SQL-TUTOR

Please confirm your choice

You have chosen to work on a different template to that suggested by the system.
Based on the current level of your knowledge of SQL, the system suggests that you work on a problem for clause HAVING clause, which is selected.
Please confirm your selection.

[✓ HAVING (System choice)]

SELECT (Your choice)

Please make your choice now and click on "Continue".

Figure 4: Confirming the user's deviation from system choice

4.3 The Introduction of Template Groups

The SQL problem templates were categorised into eight template groups. The reasons for creating template groups are given below.

SQL statements consist of up to six clauses; SQL also allows for nested statements. When learning SQL, students generally learn when each clause needs to be applied. Thus, it can be beneficial to allow students to select problems based on clauses. A higher abstraction of their student model can also be viewed based on their knowledge of SQL clauses. The problem selected by the system (system choice) could also be based on clause knowledge. This was implemented in the original version.

The final experimental version comprised of 38 problem templates. To keep the similarity between the two versions, on problem selection, students of the experimental version would have to first choose a template. This would mean making a choice from 38 templates in the experimental version, as opposed to six clauses in the control version. These numbers are dissimilar not only in the amount of choice, but also in what can be contained in STM for adequate processing (approximately 7±2 chunks). In the control group users could have all the clauses in STM, whereas with an STM capability of nine (the upper limit of the STM) it would require at least five (9 X 5 = 45) transfers between LTM and STM to think of all the templates once. Due to these reasons, templates were grouped into higher-level abstractions called template groups. Internal pedagogical decisions and modelling were based on templates.

Examples of template groups are shown in Appendices.

4.4 Preference of System Choice

For precise comparisons, all problem selection would ideally be done using only system choice for problem selection. This would ensure that, as much as possible, learning was dependent on the decisions made within the system. However, this raises two issues. First, users would be very constrained with no choice in selecting their problems. Although this restriction could be advantageous for novices, it could be frustrating and de-motivating as expertise increases. Second, it provides an artificial learning environment, where the user has no choice in what they are presented with to learn; whereas, this research is aimed at real learning situations. Due to these reasons, the decision was made to give the users choice when selecting problems, but providing motivation to accept the system choice.

Motivation to remain with system choice was done as follows. When selecting a problem, users were initially given a choice in the form of clauses (in the control group) or template-groups (in the experimental group) with the system choice selected. If users accepted the system choice, they were presented with a range of appropriate problems, with the most ideal problem highlighted. If users made a selection other than system choice, a dialog screen was presented asking for confirmation of their deviation from system choice (see Figure 4). In this dialog screen, the system choice and user choice were presented, with both choices clearly identified and the system choice selected. Once users made their choice, they were presented with the range of appropriate problems as above. See Figure 5 for the sequence followed by users in both the versions. The screenshots for this sequence are shown in *Appendix B: Selecting Problems in the Experimental Version*.

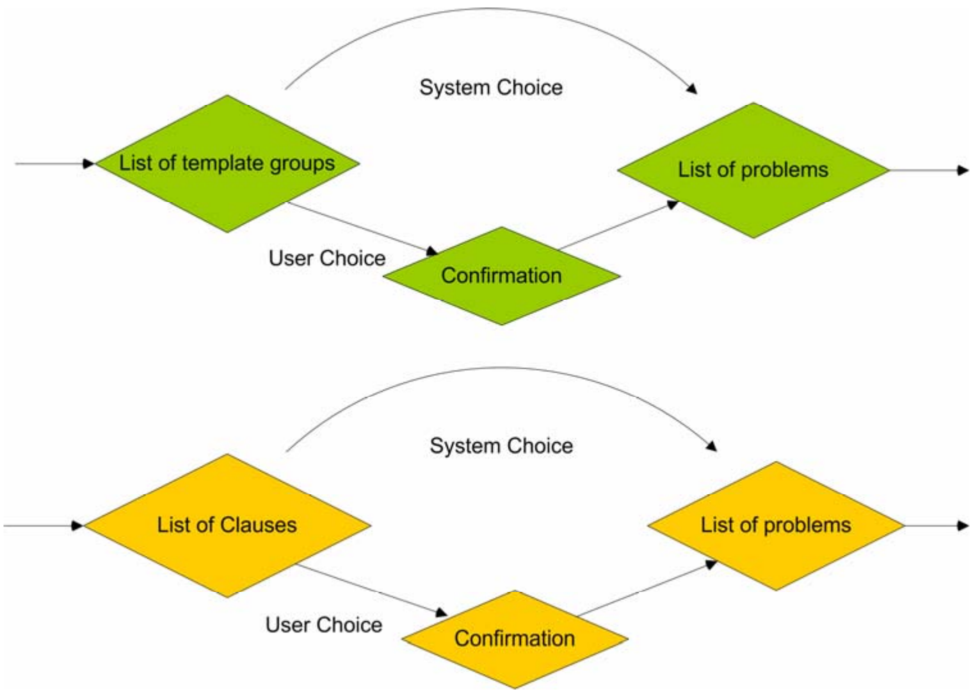


Figure 5: The problem selection process in the experimental version (top) and control version (below)

4.5 Problem Difficulty Level

In SQL-Tutor, each problem is assigned a problem difficulty level ranging from one to nine; one being the easiest and nine being the most difficult. This difficulty level is assigned by the problem author.

After creating templates, the difficulty levels of problems were compared within each template. The hypothesis here was that as templates signified a particular type of solution, they should also have the same difficulty levels. This was found to be true, and all templates fell within ±1 difficulty level of each other. An average difficulty level was then calculated for each template, and this was used in certain pedagogical decisions in the experimental version.

4.6 Student Modelling

The control version utilised the original SQL-Tutor student modelling approach. CBM was used for short-term student modelling. Long-term models contained constraint histories, knowledge scores (for each constraint), and solved problems. Templates were used for short-term student modelling in the experimental version. For long-term modelling, template histories, knowledge scores (for each template), and solved problems were stored.

In addition to the template-based models, the experimental version also stored the same long-term models as the control group version. Although these were not used in pedagogical decisions, there were maintained for analysis purposes. Comparisons between learning curves from different domain data set sizes or domain models could confound results [69].

4.7 Student's Level of Mastery

In this research, the levels of domain mastery were calculated using two separate measures: the knowledge scores and the overall knowledge level. We defined knowledge scores as a fine-grained statistic, measuring the student's mastery of the knowledge components (such as constraints in the control version, and templates in the experimental version). The overall knowledge level gave an indication to the student's mastery within the domain as a whole.

No theories exist for calculating knowledge levels or scores in an ITS. After several trials using various heuristics to calculate knowledge scores, a simple heuristic based on probabilities that has been used previously, was employed. See *Appendix C: Calculating Knowledge Scores* for an example of two such heuristics trialled. The knowledge scores and knowledge level govern the speed of progress within the course material. Two measures were used to choose the *best* heuristic. First, a subjective measure that assessed which heuristic provided sufficient time with each knowledge-component whilst still allowing the student to progress at an adequate rate. Second, the heuristic of choice had to have the least number of variables or assumptions. For instance, Heuristic A [See Section 8.3.1] has many variables and assumptions that have not been tested in prior evaluations. These variables would add to the *unknowns* that could influence the results in the evaluation.

For each version, we used a similar method of calculating the student's knowledge scores [See Heuristic B in Section 8.3.2]; for each constraint in the control version and for each template in the experimental version. All constraints (or templates) were initially given a probability of zero (i.e. no previous knowledge was assumed). If each constraint or template was used successfully twice in a row, it was assumed to be learned i.e. a knowledge score of one was assigned to it. Otherwise, an average of the last five attempts was used as an estimate of the student's knowledge.

An overall knowledge level (between one and nine) was also calculated using a similar heuristic, for each student based on the difficulty level of problems that they solved. This heuristic has also been successfully used previously.

Both student's knowledge scores and knowledge levels could fluctuate as the student progressed through the training sequence. We believe that this allows somewhat for the fluctuating nature in which students answer questions.

4.8 Problem Selection

Problem selection for system choice was based on the problem difficulty level (or template difficulty level), the student's knowledge scores, and the student's knowledge level. The problem selection was done in two steps. The strategy implemented for the experimental version was based on the strategy used in the original version of SQL-Tutor.

First, templates were partitioned into two; the partition was based on template difficulty levels. The threshold for partition was made at a template difficulty level of five. When calculating system choice, only problems from the first partition were accessible to students with knowledge levels below a certain threshold (in our case 4). All problems were included in calculating system choice for students with higher knowledge levels.

Second, from the partition selected in the first step, the template with the lowest knowledge score in order of difficulty was selected. The unsolved problems from this template were then displayed to the student.

4.9 Feedback

All six levels of feedback (simple feedback, error flag, hint, partial solution, list all errors, and full solution) were available to the student. In addition to the feedback based on constraints, participants in the experimental version also received feedback messages associated with templates for any feedback level greater than *error flag*. The feedback messages also included the generic template in text form, as included in the particular template. See *Appendices* for the format of the templates.

4.10 Tools Used

SQL-Tutor is written in Allegro Common Lisp[12] and is run on the AllegroServe web server, housed on the ICTG workstation running the Microsoft Windows XP operating system. As such, all development for this research was implemented in LISP within this environment. The Integrated Development Environment (IDE) used to create, modify, and run the application was the Allegro CL Dynamic Object-Oriented Programming System[13]. Templates and template-groups were stored as lists in files (see *Appendices* for the format of the templates and template-groups), and loaded into memory as objects on initialisation of SQL-Tutor. For both versions, student models based on constraints and logs that recorded student actions and ITS decisions were stored in files on the server. In addition to these, the experimental version also stored student models based on templates.

[12] Franz Inc: http://www.franz.com
[13] Allegro CL IDE Version 8.0, Available from Franz Inc at http://www.franz.com

5 Evaluation

5.1 Description

An evaluation study was conducted at the University of Canterbury Computer Science and Software Engineering Department[14] during the second term of 2006. The ITS used to conduct this evaluation study was SQL-Tutor [19]. At the University of Canterbury, students are taught SQL in a 200-level database course offered within the Computer Science and Software Engineering undergraduate degree structure. As part of the course, students were required to attend weekly lectures where they were taught the relevant theory. Amongst other tools, students could use SQL-Tutor to practice their skills in either the weekly lab sessions or at their leisure.

5.2 Participants

The participants for the evaluation study were students from COSC 226[15], an undergraduate database course at the University of Canterbury. Participation in the study was voluntary. Sixty-eight students enrolled in the course. Of these students, 48 logged into SQL-Tutor and completed the pre-test. Participants comprised two groups: the control and the experimental.

5.3 Method

Participants were allowed to log into SQL-Tutor during a scheduled laboratory session in the second week of May 2006. The login module within SQL-Tutor assigned students to either the control group or the experimental group; each student remained in the assigned group for the entire evaluation period.

At first login, users were presented with one of two online pre-test questionnaires. On completion of the pre-test, users continued onto their first problem. On a pre-specified date approaching the end of the evaluation period (5th June 2006), students were presented with one of two online post-test questionnaires. These were automatically administered to anyone that logged in after that particular date. In both the pre and post-tests, the questionnaires were randomly chosen by the system. The pre and post-tests used were tests created and used previously in SQL-Tutor. Each test (pre and post) contained four multi-choice questions. The pre and post-tests were of comparable complexities. The marks for the pre and post-tests were recorded.

As students used the system, their constraint and problem histories were recorded in individual student models. Student actions and any decisions made by the ITS

[14] CSSE Department, UC: http://www.cosc.canterbury.ac.nz
[15] COSC 226 information available at http://www.cosc.canterbury.ac.nz/open/teaching/

were also recorded in individual student logs. For the experimental group, the history and usage of templates was also accounted in individual template model files.

When selecting a new problem, users in the control group were first presented with a choice of SQL clauses, while users in the experimental group were presented with a choice of template groups as discussed in Section 4.4. In each case, the system choice was selected by default. If users selected a choice other than the system choice, they were presented with a confirmation dialogue, alerting them to their deviation from the system choice. Next, users were presented with the set of problems appropriate to their selection and their knowledge level. In this set, the system problem choice was highlighted; this choice could also be overridden by the user. After selecting the problem, the problem text with solution area was presented to the user. In both versions, the user could submit the solution at any stage. Six levels of feedback, ranging from simple feedback to full solution, were available to the user.

5.4 Results

5.4.1 Pre and Post-tests

Forty-eight students logged into SQL-Tutor and completed the pre-test. Of the 48 students, 23 were randomly assigned to the control group, and 25 to the experimental group. Both the pre and post-tests contained four problems, each worth one mark; thus students could gain a maximum of four marks for either of the tests. The control group had a mean of 0.5 with standard deviation of 0.07. The experimental group had a mean of 0.42 with standard deviation of 0.06. A t-Test for independent means was carried out. There was no significant difference between the control and experimental pre-test results. This means that the two groups belonged to the same population, allowing for direct comparison. Table 1 contains the analysis of the pre-test scores.

Pre-test	***Control***	***Experimental***
No. of participants	23	25
Mean	50%	42%
Standard deviation	0.07	0.06

Table 1: Analysis of pre-test results

Sixteen students completed the post-test questionnaire. However, six of these students had not attempted any problems between the pre-test and post-test, and we have not taken their post-test results into consideration. Hence, the results of the remaining ten students' pre and post-test results are given in Table 2. There was no significant difference between the control and experimental group. Furthermore, the number of students in the matched condition was too low for adequate comparisons.

Group	***No. of participants***	***Pre-test***	***Post-test***
Experimental	6	38% (0.25)	62.5% (0.2)
Control	4	42% (0.30)	62.5% (0.12)

Table 2: Analysis of matched pre and post-test results

5.4.2 Learning Curves

One method of analysing whether the objects measured are related to the concepts learned is to plot learning curves [70]. For this, the number of times the object (in our case, the constraint) is relevant was plotted against the proportion of times it was used incorrectly. If the object measured is being learned, a power curve should result (cited in [70]). Learning curves indicate whether the concepts taught in the domain have been learned. For more information on learning curves, see [69, 71].

Figure 6 shows the learning curve for the control group, while Figure 7 shows it for the experimental group. The power curve equation for the control group was $y=0.0995x^{-0.384}$, and for the experimental group it was $y=0.1044x^{-0.4079}$.

Both graphs have a good fit to the power curve (0.92 for the control, 0.89 for the experimental). The learning rate (as seen by the slope) is slightly higher for the experimental group, but not significantly.

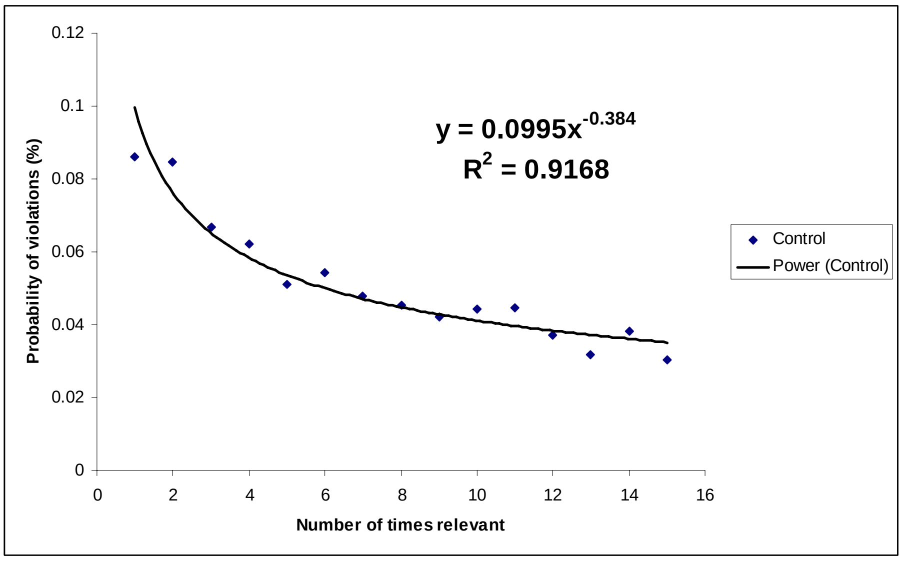


Figure 6: The probability of constraint violations for the control group

A learning curve was also plotted for templates (i.e. for the experimental group). This is shown in Figure 8. A trend line was added to this. However, the fit to the trend line was very poor ($R^2 = 0.0072$). It also does not fit to a power curve. This implies that templates were not learned by the experimental group.

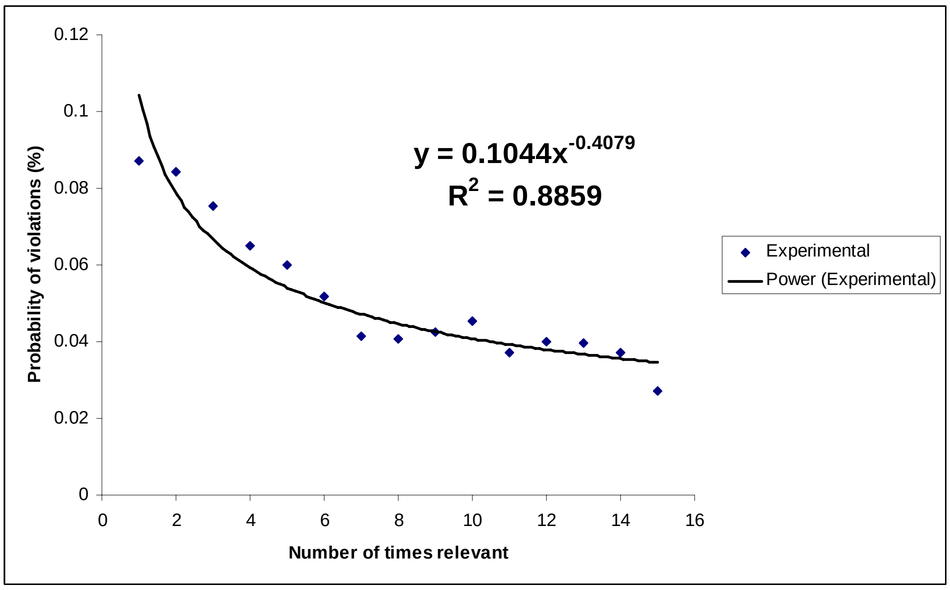


Figure 7: The probability of constraint violations for the experimental group

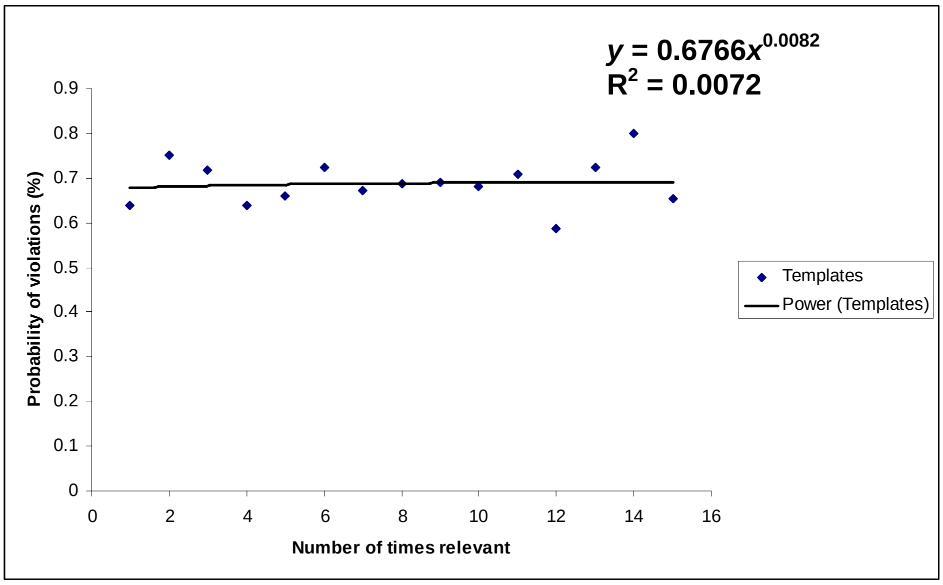


Figure 8: The probability of template violations for the experimental group

5.5 Discussion

In this evaluation, we were able to create physical representations of problem templates in the SQL domain. Templates are generally created by many experts, after years of practice in their domain of expertise. For this evaluation however, problem templates were created by recursively dividing the problem set in the SQL-Tutor domain module using certain heuristics that we created. The 38 templates were then saved in a text file and loaded into computer memory as objects when SQL-Tutor was initialised. Students in the experimental group were modelled using templates. Template histories and the knowledge scores were recorded in the long-term student model. Problem selection in the experimental group was done using problem templates. Problem templates were also used to model the students' knowledge. Knowledge scores were calculated and updated. Long-term student models containing template histories were kept, giving the ability to make adaptive pedagogical decisions. Feedback regarding templates was successfully associated to individual problem templates and presented to the students as required. Figure 2 shows the feedback given to participants in the control group, based on constraints; Figure 12 shows the specific template feedback displayed to the experimental group. The control group used constraints to model students and make pedagogical decisions such as problem selection and feedback. What is very interesting is that although the experimental group had problem selection based on templates, template-based student modelling, and template-specific feedback, they learned constraints (see the learning curves above). This means that template based methods are effective in learning domain-specific information.

Learning templates and associating the correct information takes time, practice, and ideally expert coaching. This is found in many domains, such as sports and OO design. In OO design, so many patterns exist that they have been placed into catalogues depending on their use. Although it is not expected that students memorise these templates, learning and practising them implicitly seems to increase domain knowledge.

Our study ran for a short period of time: only part of a course term. Furthermore, students did not tend to use the system at the same rate for the entire period; instead, there was a much higher level of usage towards the end of the study (before their exams). Generally, students require a large amount of time to learn, compile, and associate mental templates in domains such as sports, OO, and driving. In spite of the short time for evaluation time, students within the experimental group had high learning rates, comparable to the control group. As expected, the learning curve of the template shows that they did not explicitly learn the templates; however, they learned domain information.

Although there is significant difference between the pre-test and post-test scores of the experimental group and not of the control group, we do not conclude from this that the experimental version is better than the control, or that the increase in learning is due to templates. This statistic has to be taken with some caution. First, the number of students that participated in the post-test was relatively low (ten students in total). Second, the pre and post-test questions ask general questions about SQL. In hindsight, we believe that they do not measure accurately the level of expertise gained through use of templates. Thirdly, the significance was at $p<.1$.

This preliminary evaluation shows promising results into the theory of learning with problem templates. Within the context of the customised ITS coaching environment, it could provide a powerful mechanism of transferring expert domain information to novices, enabling them to reach much higher levels of expertise in shorter amounts of time.

6 Conclusions and Further Work

Learning affects nearly all aspects of our life. As such, there is much interest in understanding the processes of learning. Restrictions, such as resource constraints do not allow for optimal learning environments in our society. ITSs fill this need by providing a customised, adaptive environment, where the student can engage in active, deliberate, and coached learning activities. A goal of ITS development is to find means of maximising the effectiveness of learning within the ITS. This goal provided the motivation for this research.

In this paper, we introduced the notion of problem templates. Problem templates are based on the various theories of expert memory described in Section 2.2. We proposed that problem templates allow experts access to vast amounts of information regarding common problem states in their domain. These templates are stored in LTM and contain information to recognise problem states. They also contain common solution strategies and information regarding the tools required to solve the problems. Associated information can be stored as pointers to other templates, as proposed by the TT. Templates can either be formed and grouped into their associated hierarchies over time (with experience in the domain), or taught.

This research had six goals as listed in Section 3.1. Five hypotheses [see Section 3.2] were also proposed. The fulfilment of these goals and their effects on our hypotheses are summarised below.

First, thorough examination of relevant, prior research was conducted to assess the possible validity of problem templates. This included a study of the literature on ITSs, learning, memory, and expertise. We also looked at examples of where we believe problem templates are used in learning within domains such as sports, driving instruction, and OO software design. The notion of templates can be supported by these theories and examples. Second, physical representations of problem templates were created in a complex, rich, and relatively open-ended domain (SQL). Ideally, these templates would have been compiled by experts in the domain; however, this resource was not available to us. Instead, problem templates were extracted from the domain module of SQL-Tutor, using the assumption that it contained embedded expert knowledge. Third, students in SQL-Tutor were modelled using templates. Knowledge scores for each template were calculated and updated as the student progressed through the training session. Fourth, pedagogical decisions such as problem selection were based on problem templates, knowledge scores, and knowledge levels. Students were able to progress adequately within their customised path of learning. Fifth, feedback associated to templates was also presented to students. This feedback contained information about the template in text form.

Lastly, an evaluation study was conducted in which participants used either a control version of SQL-Tutor (using only constraints), or the experimental version (using templates).

The first, second, and third hypotheses were supported in this research. The validity of templates as a useful learning mechanism was demonstrated by background research, examples, and our evaluation. Templates were compiled into physical representations, and pedagogical strategies such as problem selection were made based on templates. Student models were also based on template representations. Feedback given was also specific to the templates. The fourth and fifth hypotheses were dependent on time, and as such, we do not have enough data to support these hypotheses. Nevertheless, even in short periods of training, domain learning occurred when templates were used, as can be seen from the learning curves.

Analysis of results, combined with the background research, showed great promise of using problem templates in learning situations. We are not certain that the templates created were ideal. This problem is beyond the scope of this research, and left to the experts within their own domains. Templates also take long periods to master. For instance, learning templates in sports requires many years of instruction transferred from expert coaches. In certain domains such as OO, due to the vast number of patterns, templates are catalogued into books, rather than to memory. However, learning to use templates still increases domain expertise. In spite of these restrictions on our research, learning curves showed that students in the evaluation study learned domain knowledge at a high rate - as high a rate as the control group. As explained by the time factor, the learning curve for the templates did not fit a power curve.

This research opens doors to further work in education, psychology, and ITSs. Two of these proposed extensions are described below.

1. An evaluation study with a longer evaluation period could be performed. Specific sets of pre and post-tests that test levels of expertise rather than just domain knowledge could be created and administered. These tests could be created by domain experts working with psychologists to evaluate amount of relevant domain knowledge available when faced with a particular problem state, methods of associating templates, and cognitive limitations on template learning (such as memory constraints).
2. In this paper, we looked at problem templates in SQL. Although examples of template use were given for other domains, we are not sure if templates could be physically compiled for all domains. Different types of domains could be studied to see if template creation has limitations on certain domains or applications. These templates then need to be modelled in a form appropriate for use by the ITS.

For centuries, we have tried to understand the complex processes by which humans learn. Current research looks at this from different perspectives, such as neuroscience, psychology (such as behavioural and cognitive models), or from a domain perspective (concepts, declarative or procedural knowledge). This gives us a basis to begin to understand the processes within the multifarious framework that govern the relationships between the learner, the domain, and the environment. It enables us to create environments (such as ITSs) that facilitate this learning process. In the very least, it helps us to think laterally about ourselves.

7 References

[1] A. J. Vander, J. H. Sherman, and D. S. Luciano, *Human Physiology: The Mechanics of Body Function*, TMH ed. New York: Tata McGraw-Hill Publishing Company Limited, 1975.

[2] L. R. Squire, "Declarative and Nondeclarative Memory: Multiple Brain Systems Supporting Learning and Memory," vol. v4, pp. p232(12), 1992.

[3] J. E. LeDoux, "Brain mechanisms of emotion and emotional learning," *Current Opinion in Neurobiology*, vol. 2, pp. 191-197, 1992.

[4] S. E. Petersen, H. v. Mier, J. A. Fiez, and M. E. Raichle, "The Effects of Practice on the Functional Anatomy of Task Performance," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 95, pp. 853-860, 1998.

[5] F. Bennett, *Computers as Tutors: Solving the Crisis in Education*: Faben Inc, 1999.

[6] R. S. Prawat, "Teachers' Beliefs about Teaching and Learning: A Constructivist Perspective," *American Journal of Education*, vol. 100, pp. 354-395, 1992.

[7] A. Mitrović, "SQL-Tutor: A Preliminary Report," Computer Science & Software Engineering Department, University of Canterbury, Christchurch, Technical Report TR-COSC 08/97, 21 August 1997.

[8] K. A. Ericsson and N. Charness, "Expert Performance; Its Structure and Acquisition," *American Psychologist*, vol. 49, pp. 725-747, 1994.

[9] J. R. Anderson, *Rules of the Mind*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1993.

[10] S. Papert, "What's the Big Idea? Toward a Pedagogy of Idea Power," *IBM Systems Journal*, vol. 39, pp. 720-729, 2000.

[11] A. Corbett, "Cognitive Computer Tutors: Solving the Two- Sigma Problem," presented at User Modeling 2001: 8th International Conference, UM 2001, LNAI 2109, Sonthofen, Germany, 2001.

[12] K. VanLehn, C. Lynch, K. Schulze, J. A. Shapiro, R. Shelby, L. Taylor, D. Treacy, A. Weinstein, and M. Wintersgill, "The Andes Physics Tutoring System: Five Years of Evaluations," in *Artificial Intelligence in Education - Supporting Learning Through Intelligent and Socially Informed Technology*, vol. 125, *Frontiers in Artificial Intelligence and Applications*, C.-K. Looi, G. McCalla, B. Bredeweg, and J. Breuker, Eds. Washington, D.C.: IOS Press, 2005, pp. 678-685.

[13] A. Mitrović and S. Ohlsson, "Evaluation of a Constraint-Based Tutor for a Database Language," *International Journal of Artificial Intelligence in Education*, vol. 10, pp. 238-256, 1999.

[14] B. S. Bloom, "The 2 Sigma Problem: The Search for Method of Group Instruction as Effective as One-to-One Tutoring," *Educational Researcher*, vol. 13, pp. 4-16, 1984.

[15] A. D. de Groot, *Thought and Choice in Chess*. The Hague, Netherlands: Mouton Publishers, 1978.

[16] F. Gobet and H. A. Simon, "Expert Chess Memory: Revisiting the Chunking Hypothesis," *Psychology Press*, vol. 6, pp. 225-255, 1998.
[17] W. G. Chase and H. A. Simon, "Perception in Chess," *Cognitive Psychology*, vol. 4, pp. 55-81, 1973.
[18] F. Gobet and H. A. Simon, "Templates in Chess Memory: A Mechanism for Recalling Several Boards," *Cognitive Psychology*, vol. 31, pp. 1-40, 1996.
[19] A. Mitrović, "An Intelligent SQL Tutor on the Web," *International Journal of Artificial Intelligence in Education*, vol. 13, pp. 173-197, 2003.
[20] S. Ohlsson, "Learning and Instruction: An Introduction," Department of Psychology, University of Illinois, Illinois, Chicago, Course Notes 2004.
[21] L. Vygotsky, *Mind in Society: The Development of Higher Psychological Processes*. Cambridge, MA: Harvard University Press, 1978.
[22] A. Mitrović and B. Martin, "Scaffolding and Fading Problem Selection in SQL-Tutor," presented at The 11th International Conference on Artificial Intelligence in Education (AIED), Sydney, 2003.
[23] J. Beck, M. Stern, and E. Haugsjaa, "Applications of AI in Education (Special Issue on Artificial Intelligence)," *Crossroads*, vol. 3, pp. 11-15, 1996.
[24] P. Holt, S. Dubs, M. Jones, and J. Greer, "The State of Student Modelling," in *Student Modelling: The Key to Individualized Knowledge-based Instruction*, vol. 125, *Computer Systems and Sciences*, J. E. Greer and G. I. McCalla, Eds. Berlin: Springer-Verlag GmbH, 1994, pp. 3-35.
[25] J. A. Self, "Bypassing the Intractable Problem of Student Modeling," in *Intelligent Tutoring Systems: at the Crossroad of Artificial Intelligence and Education*, C. Frasson and G. Gauthier, Eds. Norwood, NJ: Ablex Publishing Corporation, 1990, pp. 107--123.
[26] K. G. Schulze, R. N. Shelby, D. J. Treacy, M. C. Wintersgill, K. VanLehn, and A. Gertner, "ANDES: An Intelligent Tutor for Classical Physics," *The Journal of Electronic Publishing*, vol. 6, 2000.
[27] A. Mitrović, "The Effect of Explaining on Learning: a Case Study with a Data Normalization Tutor," in *Artificial Intelligence in Education - Supporting Learning through Intelligent and Socially Informed Technology*, vol. 125, *Frontiers in Artificial Intelligence and Applications*. Washington, D.C.: IOS Press, 2005, pp. 499-505.
[28] A. S. Gertner and K. VanLehn, "Andes: A Coached Problem Solving Environment for Physics," in *Intelligent Tutoring Systems*, vol. 1839, *Lecture Notes in Computer Science*, G. Gauthier, C. Frasson, and K. VanLehn, Eds. Berlin: Springer-Verlag, 2000, pp. 133-142.
[29] A. Mitrović, "NORMIT: A Web-Enabled Tutor for Database Normalization " presented at 2002 International Conference on Computers in Education (ICCE'02), Auckland, 2002.
[30] R. E. Mayer, *Mulltimedia Learning*, 7 ed. New York: Cambridge University Press, 2005.
[31] A. Mitrović, K. R. Koedinger, and B. Martin, "A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modeling," presented at 9th International Conference User Modeling 2003, UM 2003, Johnstown, PA, USA, 2003.
[32] J. R. Anderson, *The Architecture of Cognition*. Cambridge, MA: Harvard University Press, 1983.
[33] J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier, "Cognitive Tutors: Lessons Learned," *The Journal of the Learning Sciences*, vol. 4, pp. 167-209, 1995.
[34] A. T. Corbett, J. R. Anderson, and A. T. O'Brien, "Student Modeling in the ACT Programming Tutor," in *Cognitively Diagnostic Assessment*, P. Nichols, S. Chipman, and B. Brennan, Eds. Hillsdale, NJ: Lawrence Erlbaum Associates, 1995, pp. 19-41.
[35] A. T. Corbett and J. R. Anderson, "Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge," *User Modeling and User-Adapted Interaction*, vol. 4, pp. 253-278, 1994.

[36] S. Ohlsson, "Constraint-Based Student Modelling," in *Student Modelling: The Key to Individualized Knowledge-based Instruction*, vol. 125, J. E. Greer and G. I. McCalla, Eds. Berlin: Springer-Verlag GmbH, 1994, pp. 167-189.

[37] S. Ohlsson, "Learning From Performance Errors," *Psychological Review*, vol. 103, pp. 241-262, 1996.

[38] A. Mitrović, M. Mayo, P. Suraweera, and B. Martin, "Constraint-based Tutors: a Success Story," presented at 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Berlin, 2001.

[39] A. Mitrović and B. Martin, "Evaluating Effectiveness of Feedback in SQL-Tutor," presented at Proceedings of the International Workshop for Advanced Learning Technologies IWALT2000, Palmerston North, 2000.

[40] A. Mitrović, B. Martin, and M. Mayo, "Using Evaluation to Shape ITS Design: Results and Experiences with SQL-Tutor," *International Journal of User Modelling and User Adapted Interaction*, vol. 12, pp. 243-279, 2002.

[41] D. C. Donald and F. B. Raymond, "SEQUEL: A Structured English Query Language," in *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*. Ann Arbor, Michigan: ACM Press, 1974.

[42] R. Elmasri and S. B. Navathe, "SQL - The Relational Database Standard," in *Fundamentals of Database Systems*, M. Suarez-Rivas and K. Harutunian, Eds., 4 ed. Reading, Massachusetts: Addison Wesley Longman Inc., 2004.

[43] A. Mitrović and B. Martin, "Evaluating Adaptive Problem Selection," in *Adaptive Hypermedia and Adaptive Web-Based Systems*, vol. 3137/2004, *Lecture Notes in Computer Science*. Berlin: Springer-Verlag GmbH, 2004, pp. 185-194.

[44] A. Mitrović and B. Martin, "Evaluating the Effects of Open Student Models on Learning," presented at 2nd International Conference on Adaptive Hypermedia and Adaptive Web-based Systems AH 2002, Malaga, Spain, 2002.

[45] D. Hartley and A. Mitrović, "Supporting Learning by Opening the Student Model," presented at 6th International Conference on Intelligent Tutoring Systems ITS 2002, Biarritz, France, 2002.

[46] D. E. Feldman and M. Brecht, "Map Plasticity in Somatosensory Cortex," *Science*, vol. 310, pp. 810-815, 2005.

[47] M. Sur and J. L. R. Rubenstein, "Patterning and Plasticity of the Cerebral Cortex," *Science*, vol. 310, pp. 805-810, 2005.

[48] A. Karni, "The acquisition of perceptual and motor skills: a memory system in the adult human cortex," *Cognitive Brain Research*, vol. 5, pp. 39-48, 1996.

[49] F. Gobet and A. J. Waters, "The Role of Constraints in Expert Memory," *Journal of Experimental Psychology*, vol. 29, pp. 1082-1094, 2003.

[50] K. A. Ericsson, W. G. Chase, and S. Faloon, "Acquisition of a Memory Skill," *Science*, vol. 208, pp. 1181-1182, 1980.

[51] F. Gobet, "Expert Memory: A Comparison of Four Theories," *Cognition*, vol. 66, pp. 115-152, 1998.

[52] K. J. Vicente and J. H. Wang, "An Ecological Theory of Expertise Effects in Memory Recall," *Psychological Review*, vol. 105, pp. 33-57, 1998.

[53] F. Gobet and G. Clarkson, "Chunks in Expert Memory: Evidence for the Magical Number Four ... Or is it Two?," *Memory*, vol. 12, pp. 732-747, 2004.

[54] K. A. Ericsson and P. G. Polson, "An Experimental Analysis of the Mechanisms of a Memory Skill," *Journal of Experimental Psychology*, vol. 14, pp. 305-316, 1988.

[55] G. A. Miller, "The Magical Number Seven, Plus or Minus Two," *The Psychological Review*, vol. 63, pp. 81-97, 1956.

[56] R. W. Engle, "Working Memory Capacity as Executive Attention," *Current Directions in Psychological Science*, vol. 11, pp. 19, 2002.

[57] N. Cowan, "The Magical Number 4 in Short-term Memory: A Reconsideration of Mental Storage Capacity," *Behavioral and Brain Sciences*, vol. 24, pp. 87-185, 2001.

[58] K. A. Ericsson and W. Kintsch, "Long Term Working Memory," *Psychological Review*, vol. 102, pp. 211-245, 1995.

[59] A. D. Baddeley and G. J. Hitch, "Working Memory," in *The Psychology of Learning and Motivation*, vol. 8, G. Bower, Ed. New York: Academic Press, 1974, pp. 47-89.

[60] F. Gobet, P. C. R. Lane, Steve Croker, P. C.-H. Cheng, G. Jones, I. Oliver, and J. M. Pine, "Chunking Mechanisms in Human Learning," *TRENDS in Cognitive Sciences*, vol. 5, pp. 236-243, 2001.

[61] J. Shanteau, "The Psychology of Experts: An Alternative View," in *Expertise and Decision Support*, G. Wright and F. Bolger, Eds. NY: Plenum Press, 1992, pp. 11-23.

[62] S. A. Ferguson, "Other High-risk Factors for Young Drivers - How Graduated Licencing Does, Doesn't or Could Address Them," *Journal of Safety Research*, vol. 34, pp. 71-77, 2003.

[63] M. E. Thomson, "Aviation Risk Perception: A comparison between Experts and Novices," *Risk Analysis: an Official Publication of the Society for Risk Analysis*, vol. 24, pp. 1585, 2004.

[64] K. A. Ericsson, R. T. Krampe, and C. Tesch-Römer, "The Role of Deliberate Practice in the Acquisition of Expert Performance," *Psychological Review*, vol. 100, pp. 363-406, 1993.

[65] C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language*: Oxford University Press, 1977.

[66] C. Alexander, *The Timeless Way of Building*: Oxford University Press, 1979.

[67] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Elements of Reusable Object-Oriented Software*. Boston, MA: Addison-Wesley, 1995.

[68] M. Fowler, *Analysis Patterns : Reusable Object Models*. Menlo Park, California: Addison Wesley, 1997.

[69] B. Martin, K. R. Koedinger, A. Mitrovic, and S. Mathan, "On Using Learning Curves to Evaluate ITS," presented at The 12th international conference on Artificial Intelligence in Education, Amsterdam, 2005.

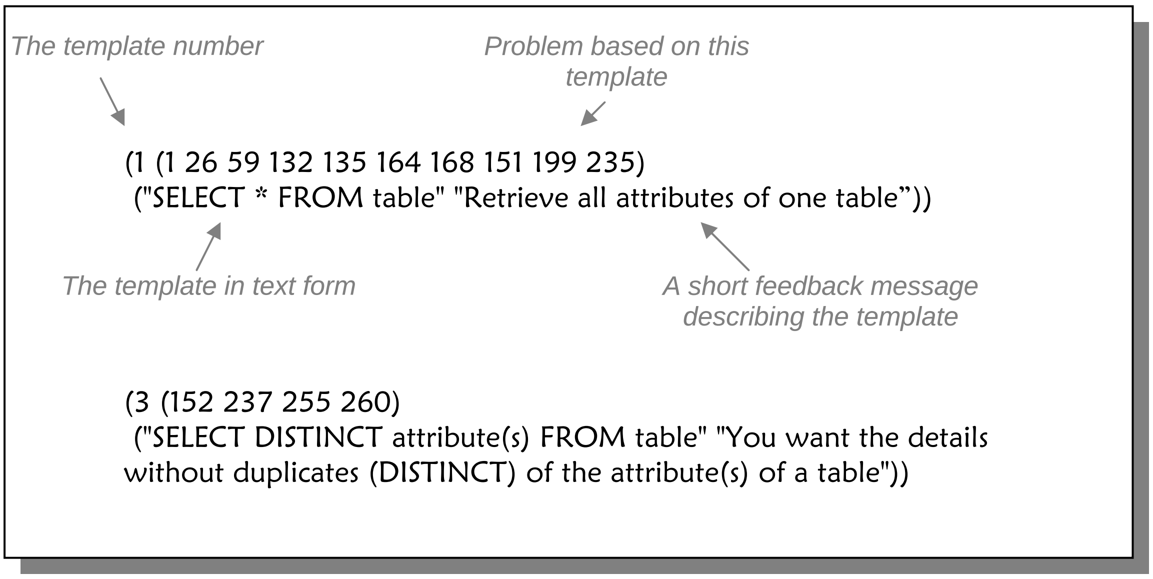
[70] B. Martin and A. Mitrović, "Using Learning Curves to Mine Student Models," presented at The 10th international conference on User Modelling UM2005, Edinburgh, 2005.

[71] B. Martin and A. Mitrović, "Evaluating Intelligent Education Systems with Learning Curves," presented at The Workshop on Evaluation at Adaptive Hypermedia 2004, Eindhoven, The Netherlands, 2004.

8 Appendices

8.1 Appendix A: SQL Problem Templates

Forty-two templates were used in the experimental version of SQL-Tutor. Two of the templates (templates 1 and 3) are given as examples below.



```
(1 (1 26 59 132 135 164 168 151 199 235)
 ("SELECT * FROM table" "Retrieve all attributes of one table”))

(3 (152 237 255 260)
 ("SELECT DISTINCT attribute(s) FROM table" "You want the details
without duplicates (DISTINCT) of the attribute(s) of a table"))
```

The template number
Problem based on this template
The template in text form
A short feedback message describing the template

Figure A.1: Two templates (template 1 and 3) used in the experimental version

As the implementation was done in LISP, each template was stored as a list of lists. The first integer in the list was the template number.

8.1.1 Template Groups

The templates are categorised into eight template groups. An example of a template group is shown in Figure A.2. Note that the two templates shown in Figure A.1 (templates 1 and 3) are part of the template group shown in Figure A.2 (template group 1).

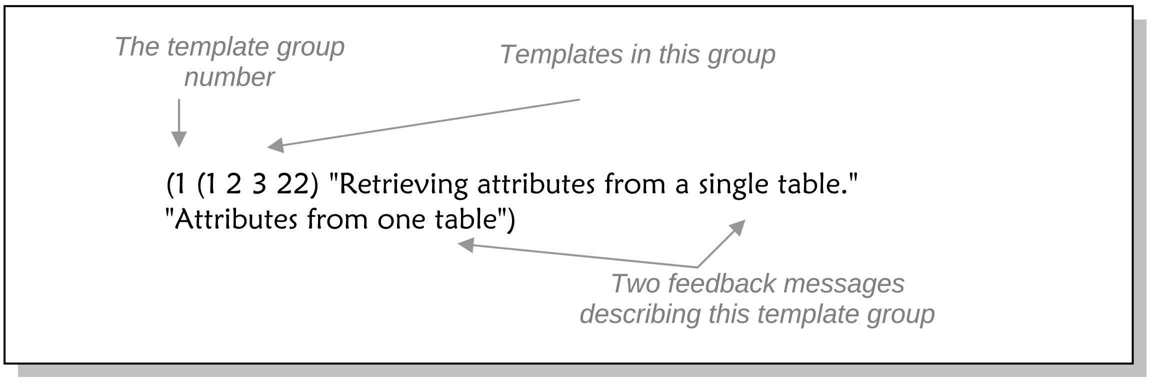
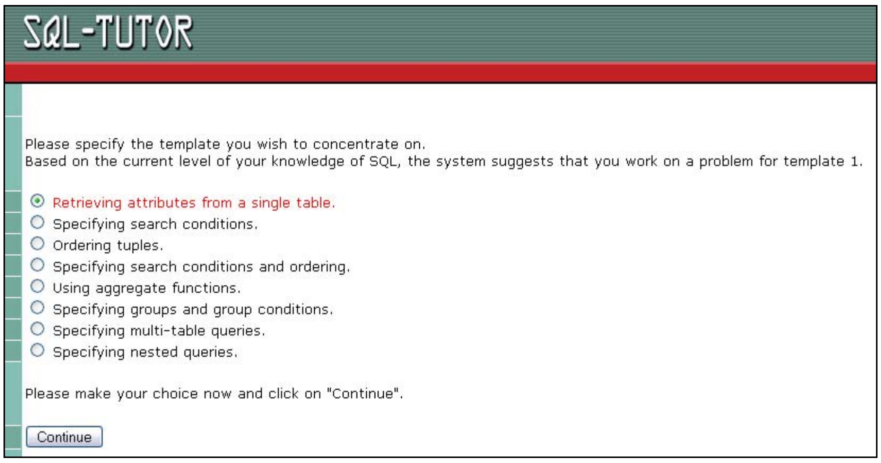

The template group number
Templates in this group
(1 (1 2 3 22) "Retrieving attributes from a single table."
"Attributes from one table")
Two feedback messages describing this template group

Figure A.2: An example of a template group used in the experimental version

8.2 Appendix B: Selecting Problems in the Experimental Version

The student is presented with the choice of template-groups, with the system choice highlighted and selected [see Figure 9].



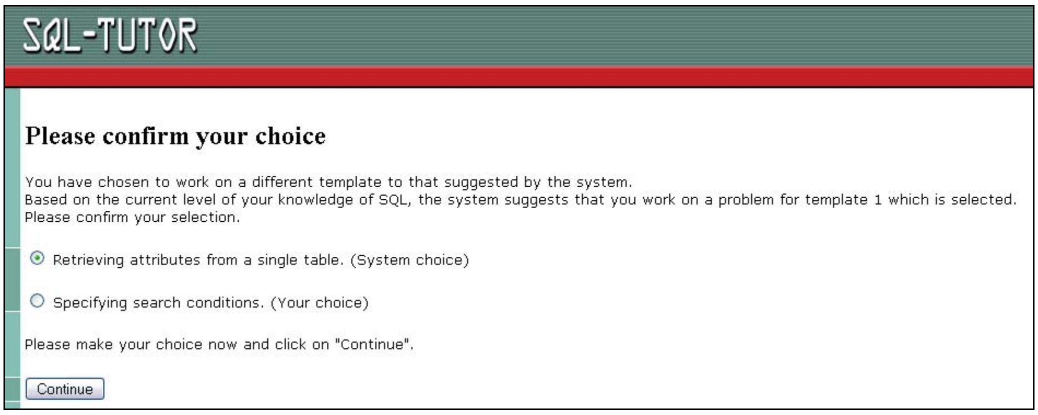
Please specify the template you wish to concentrate on.
Based on the current level of your knowledge of SQL, the system suggests that you work on a problem for template 1.

[✓ Retrieving attributes from a single table.]
Specifying search conditions.
Ordering tuples.
Specifying search conditions and ordering.
Using aggregate functions.
Specifying groups and group conditions.
Specifying multi-table queries.
Specifying nested queries.

Please make your choice now and click on "Continue".

Figure 9: Template-group choice in the experimental version

If the student selects a problem other than the system choice, a confirmation screen is displayed [see Figure 10].



Please confirm your choice

You have chosen to work on a different template to that suggested by the system.
Based on the current level of your knowledge of SQL, the system suggests that you work on a problem for template 1 which is selected.
Please confirm your selection.

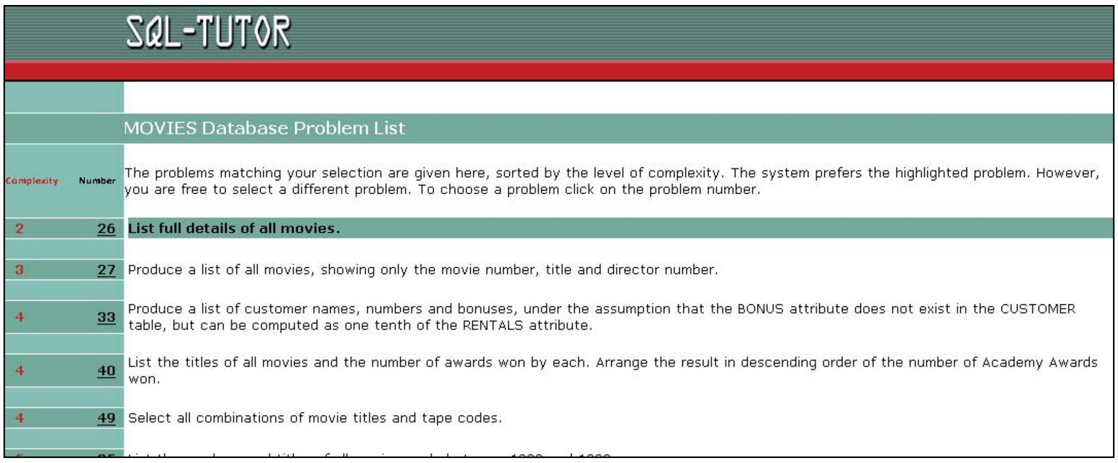
[✓ Retrieving attributes from a single table. (System choice)]

Specifying search conditions. (Your choice)

Please make your choice now and click on "Continue".

Figure 10: The confirmation dialog screen

Once the student chooses the template, a list of problems in the template is displayed with the system choice highlighted [See Figure 11].



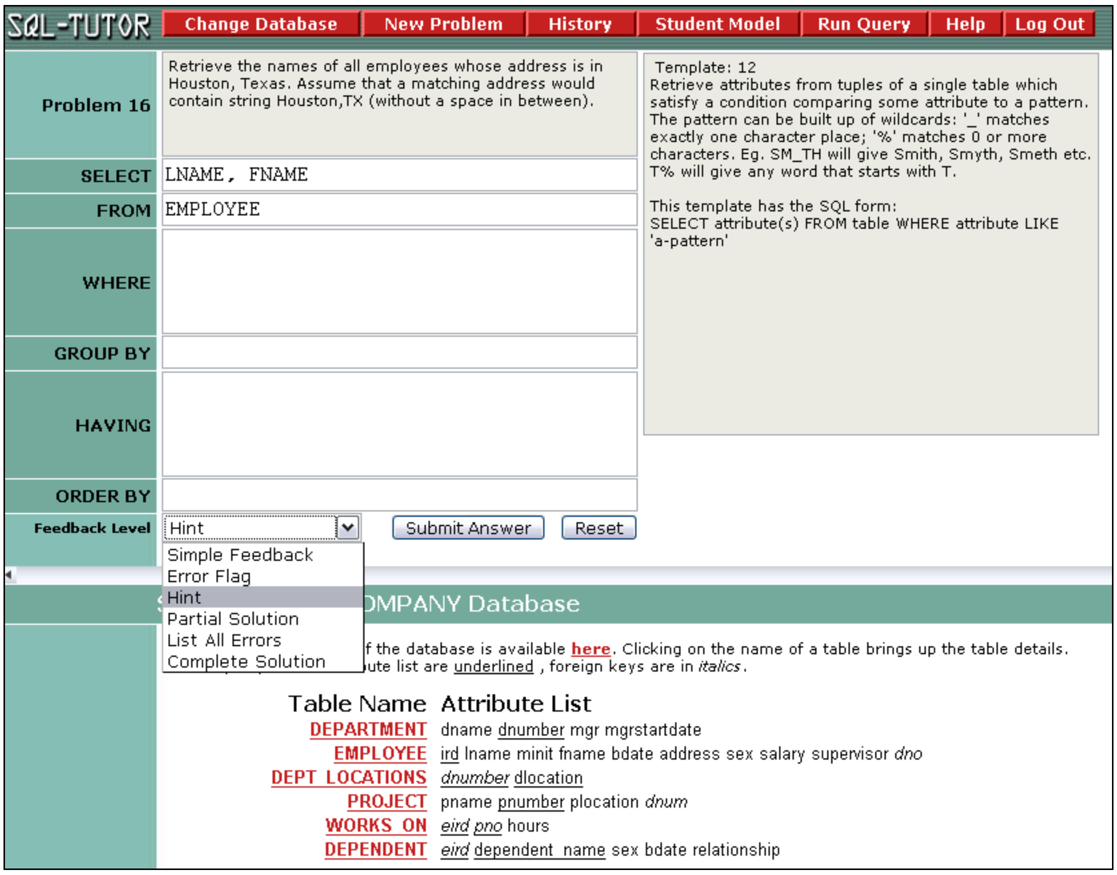
MOVIES Database Problem List

The problems matching your selection are given here, sorted by the level of complexity. The system prefers the highlighted problem. However, you are free to select a different problem. To choose a problem click on the problem number.

Complexity	Number	
2	26	**List full details of all movies.**
3	27	Produce a list of all movies, showing only the movie number, title and director number.
4	33	Produce a list of customer names, numbers and bonuses, under the assumption that the BONUS attribute does not exist in the CUSTOMER table, but can be computed as one tenth of the RENTALS attribute.
4	40	List the titles of all movies and the number of awards won by each. Arrange the result in descending order of the number of Academy Awards won.
4	49	Select all combinations of movie titles and tape codes.

Figure 11: The problem list, with the system choice highlighted

After making a selection from the problem list, the problem, solution space, feedback area, and associated information are displayed in the task environment [see Figure 12]. The student can attempt to solve the problem, request feedback, request, help, view their student model, view their history, or run their query on a sample database.



Problem 16

Retrieve the names of all employees whose address is in Houston, Texas. Assume that a matching address would contain string Houston,TX (without a space in between).

Template: 12
Retrieve attributes from tuples of a single table which satisfy a condition comparing some attribute to a pattern. The pattern can be built up of wildcards: '_' matches exactly one character place; '%' matches 0 or more characters. Eg. SM_TH will give Smith, Smyth, Smeth etc. T% will give any word that starts with T.

This template has the SQL form:
SELECT attribute(s) FROM table WHERE attribute LIKE 'a-pattern'

SELECT	LNAME, FNAME
FROM	EMPLOYEE
WHERE	
GROUP BY	
HAVING	
ORDER BY	
Feedback Level	Hint

MPANY Database

f the database is available **here**. Clicking on the name of a table brings up the table details.
ute list are underlined , foreign keys are in *italics*.

Table Name	Attribute List
DEPARTMENT	dname dnumber mgr mgrstartdate
EMPLOYEE	ird lname minit fname bdate address sex salary supervisor *dno*
DEPT_LOCATIONS	*dnumber* dlocation
PROJECT	pname pnumber plocation *dnum*
WORKS_ON	*eird* *pno* hours
DEPENDENT	*eird* dependent_name sex bdate relationship

Figure 12: The task environment in the experimental group

The control group undergoes the same sequence for problem choice. However, they choose from a set of clauses, rather than template-groups.

8.3 Appendix C: Calculating Knowledge Scores

Various heuristics for calculating the knowledge scores were created and trialled before one was implemented. Below are descriptions of two heuristics. The first is an example of a heuristic we created but did not use in the evaluation. The second is one that has previously been employed in SQL-Tutor, which we used in the final implementation. Constraint histories given below consist of zeros (constraint was violated) and ones (constraint was used correctly).

8.3.1 Heuristic A: The Three Phases of Knowledge-component Mastery

After studying constraint histories of previous long-term student models, we proposed that the path to knowledge-component mastery approximated three phases: *initial learning, mastering, and proficiency*.

Student constraint histories seemed to approximate an ideal model shown below:

0 0 0 1 (a number of 0s followed by a 1)

0 1 1 0 1 0 (a number of 0s and 1s)

1 1 1 1 (a number of only 1s)

From this assumption, an algorithm was created to calculate a difficulty level (how difficult the student found this constraint to be) for each constraint ranging from one (very easy) to nine (very difficult). The algorithm consisted of three steps:

1. The histories were divided into the three phases. The first phase comprised of all the attempts up to and including the first correct use of the constraint. Then the third phase was found, by selecting all the 1s from the end of the history. The period in between phase one and phase three was termed the learning phase. This phase consists of a number of zeros and ones.
2. The proportion of 0s in each phase was multiplied by a *phase constant* to give the *phase fraction*. For the last phase, a window size of five was assumed. This guaranteed the change from mastering to proficiency. The phase fraction is each phase's contribution to the final difficulty level of the constraint.
3. The phase levels were added and normalised to fit the 1-9 scale of difficulty.

8.3.1.1 Examples

Example 1:

Assuming a phase constant of 1/3:

Constraint history: 0 0 0 1 : 0 1 1 0 1 0 : 1 1 1 1 1

The three phases are separated by colons in the history given above.

(3/4 X 1/3) + (1/2 X 1/3) + (0/5 X 1/3) = 5/12

After normalising to a 1-9 scale, difficulty of this constraint = 4.3

Example 2:

Assuming a phase constant of 1/3

Constraint history: 1: 1: 1 1 1 1 1 1

The three phases are separated by colons in the history given above.

(0 X 1/3) + (0 X 1/3) + (0 X 1/3) = 0

After normalising to a 1-9 scale, difficulty of this constraint = 1

Example 3:

Assuming a phase constant of 1/3:

Constraint history: 0 0 0 0 0 1 : 0 1 0 1 0 : 1 1 1

The three phases are separated by colons in the history given above.

(5/6 X 1/3) + (3/5 X 1/3) + (2/5 X 1/3) = 33/54

After normalising to a 1-9 scale, difficulty of this constraint = 6

8.3.1.2 Limitations of Heuristic A

This heuristic seems to work relatively well with static and complete data i.e. post analysis. However, it was very difficult to analyse students dynamically as they learned i.e. "*Which stage are they in now?*"

We assumed a phase constant of 1/3. In other words, we implied that each phase contributed equally to the difficulty level of the constraint. Further studies using large amounts of data would have to be done to determine a more accurate figure.

We assumed that the third stage of learning only contains ones. This meant that it was very difficult for a user to be fully proficient at a knowledge-constraint.

A window size of five was required for phase three, further restricting the definition of proficiency. This window size was chosen as it has been previously used in other heuristics successfully. Does a student have to have at least five correct attempts to be proficient at a knowledge component?

How this difficulty level relates to pedagogical decisions needs to be further explored. When making pedagogical decisions do we always include all the phases, or just the most current phase? Does maintaining the initial phases in the score give a sense of how *quickly* a student mastered the knowledge-component, and therefore aid in future pedagogical decisions, or is that information irrelevant?

8.3.2 Heuristic B: Dynamic Knowledge Score

This heuristic has previously been used successfully in SQL-Tutor. This heuristic enlists the use of a floating window to ascertain the relevant section of the student's knowledge-component history to enable pedagogical decisions based on the student's current knowledge level. The result of this calculation gives the knowledge score as a probability (0-1) that a knowledge-component has been learnt.

The algorithm used for this heuristic is simple. Each knowledge-component has a default knowledge score of zero (i.e. no knowledge is assumed initially). The window of relevant knowledge-components always contains the most current attempts. Initially a window size of two (the last two attempts) is viewed. If both the attempts were successful (i.e. two 1s), then the knowledge-component is said to be learned i.e. it has a knowledge score of one. If not, the window size is increased to contain the last five attempts, and an average value is taken. This average gives the knowledge score.

The advantage of this heuristic is that it can be easily used in a dynamic environment, where the student models are continually changing. As the student progresses through the training sequence, the floating window moves to accommodate the student's current knowledge.

8.3.2.1 Examples

Given a knowledge-component history of (0 0 1 1 0 1 0 1 1), the knowledge score would be 1, as the last two attempts were successful.

Given a knowledge-component history of (0 0 0 1 0 0 1 1 0), the knowledge score would be 2/5, an average of the last five attempts.